Computing solution concepts in games with integer decisions

by

Christopher Thomas Ryan

B.A., The University of British Columbia, 2005

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES

(Business Adminstration)

THE UNIVERSITY OF BRITISH COLUMBIA

(Vancouver)

July, 2010

Abstract

Computing solution concepts in games is an important endeavor in the field of algorithmic game theory. This thesis explores two contexts – simultaneous games and sequential games – for computing solution concepts in games where some or all of the decisions of players can be modeled as integer vectors.

In the case of simultaneous games, the problem of computing pure strategy in Nash equilibrium (pure NE) is considered in two classes of *compactly represented games*; that is, games where the set of players, actions or utilities are expressed *compactly* by an input smaller than the standard *normal form* representation. The first class considered is *integer programming games* where actions are sets of integer points and utilities have a special *difference of piecewise linear convex* structure. The second class considered is symmetric games with general piecewise linear utilities. Both classes can be highly expressive of general normal form games, but can also be highly compact. The main results are polynomial-time algorithms for counting and finding pure NE when certain parameters are fixed and where the input is a description of the piecewise linear utilities. In both cases this can yield improved efficiency over known algorithms.

In the case of sequential games, the problem of finding optimal "optimistic solutions" to bilevel integer programs are explored. In particular by using recent results from parametric integer programming, a polynomial-time (in fixed dimension) algorithm is derived. Existence results for optima in a more general setting involving a multilinear lower level objective representing leader-controlled units costs are also stated using a decomposition of the original

problem into a finite set of simpler subproblems. This decomposition theorem is derived from an algebraic theory of integer programming using Gröbner bases.

Another contribution of this work is the use of a theory of encoding lattice points in polytopes as rational generating functions to encode sets of pure equilibria and bilevel optimal solutions, which are viewed as lattice point sets. This encoding then yields efficient algorithms to count, enumerate and optimize over our sets of interest.

Preface

This preface provides a statement of co-authorship for the work contained in this thesis.

The work found in Chapter 2 was undertaken with Maurice Queyranne and Matthias Köppe. I played a key role in identifying and designing the research questions of the chapter, based on some preliminary results on using rational generating functions to compute equilibria by Matthias Köppe. A version of the chapter was prepared by myself and submitted for publication. The most general statement of the results, including both convex and concave pieces, was derived by myself. Two simpler cases were derived collaboratively with Queyranne and Köppe. Several of the extensions found in the chapter were designed and derived independently. The chapter in this thesis provides additional results, a numerical example, and a rewritten introduction and conclusion.

The work found in Chapter 3 was undertaken with Albert Xin Jiang and Kevin Leyton-Brown. A conference version of this chapter will appear in the Proceedings of the Tenth Annual ACM conference on Electronic Commerce (2010). Jiang and I identified and designed the research in this chapter collaboratively. The main results were derived in a series of weekly discussions with Jiang, and I worked out several of the extensions independently. I prepared a large portion of the manuscript, whereas Sections 3.2.1 was first drafted by Jiang, which I adapted for this thesis. Section 3.6 was prepared collaboratively with Jiang for the conference version, which I adapted for the thesis.

The work found in Chapter 4 was undertaken by Matthias Köppe and Maurice Queyranne. A version of the paper recently appeared in the July 2010 issue (Volume 146, Number 1) of

the *Journal of Optimization Theory and Applications* on pages 137–150. The initial direction was proposed by Köppe and elaborated in joint collaborative research with Köppe and Queyranne. The content of Chapter 4 was prepared collaboratively with Maurice Queyranne. The additional section on generating functions methods (Section 4.5) was prepared independently and is not found in the journal version.

The work in Chapter 5 was identified by myself, and the research program was undertaken collaboratively with Maurice Queyranne and later Matthias Köppe. Many of the proofs were tackled collaboratively with Queyranne, while some were undertaken independently. I prepared the content of the chapter as it is appears in this thesis.

Table of Contents

List of Tables

List of Figures

Acknowledgments

Writing a document like this one is a journey filled with many mountains to climb and rivers to cross and many helping hands along the way. I share these words of gratitude as a small token of my appreciation.

To my advisor Prof. Maurice Queyranne, I am indebted to his patience, powerful sense of curiosity, care for rigor and joy for solving problems. I am thankful for his faith in me and my abilities and his support in traveling to meeting others with interest in the topics presented here. He is a most encouraging teacher and collaborator, and his guidance in helping me develop this line of research is undoubtedly the key ingredient to my achievements.

To my co-author Prof. Matthias Köppe and I am thankful for his welcoming acceptance of my ideas and efforts, and his drive to explore the boundaries of what is known, as is reflected here. He is a wonderful collaborator and teacher and I owe many insights to his patient explanations.

I also want to thank Mahesh Nagarajan for his tireless support and belief in me and in my work. I relied on him many times for advice and wisdom in maneuvering the strange world we call academia. Thanks also to Tom McCormick and Stephanie Van Willigenburg for serving on my research committee.

To my friend and co-author Albert Xin Jiang I want to honor his open and inquisitive mind. The chapter on symmetric games is a product of weekly meetings we had to discuss a variety of directions and ideas, and the experience has been one of the most fruitful intellectual endeavors in my short career as an academic. Thanks also goes to Prof. Kevin

Leyton-Brown for his advice on guidance on developing this research, not to mention his openness in allowing me to participate in the activities of his research group.

I also thank Prof. Dr. Robert Weismantel for supporting me to visit the Otto Von Guericke Universität Magdeburg in Magdeburg, Germany in May 2008. I am also indebted to Profs. Jesús de Loera and Rekha Thomas who were so kind in corresponding with me as I was learning about the tools used in this thesis. I also cannot forget the contribution of Prof. Richard Anstee, who believed in me at an early stage in my education and has been a friend and confidant ever since.

I am also grateful for the financial support of the National Science and Engineering Research Council and the family of Shelby L. Brumelle.

Last but certainly not least, is the love and support of my friends and family. I want to acknowledge my Uncle Bruce and Aunt Nancy, who helped me in my early days. To my Aunt Lynne, Uncle Wayne and Dave I owe my enduring sense of calm and security during my time in Vancouver and having always had a warm home-cooked meal during some cold times. To my parents, all I can say is that their love is the air I breathe, and could not have imagined getting to where I am without them. My father inspired my love of learning since the time I can remember, and my mother is like an endless fountain of joy that quenches the thirst of my heart. My gratitude also goes out to my grandmother, my great champion and inspiration. And to my brother, to whom I dedicate this work, my fellow soldier in the battle of life; the undaunted hero securing the beachhead for all, including myself, to pass.

My last word goes to the "co-author" of this work and my life during my time as a PhD student: my wife and confidant, Cecilia Jieren Wang. She is the moon that lights the way in my dark nights, the soft yellow flower that brightens my days. These closing lines are for her:

The world is hope -
Long walks in old shoes,

With garlic-stained fingers

And shared laughs

And shared tears.

The world is love,

Made alive in the spirit.

What an amazing thing to share:

Thanks, thanks, thanks.

Dedication

To my brother, Paul.

Chapter 1

Introduction

Understanding how groups of decision-makers make decisions when the outcomes they face depend intimately on one another's choices is a fundamental problem which researchers in a variety of fields have taken on as a challenge for well over a century. Such scenarios have come to be known as games and the subject which studies them as game theory.

The field attained maturity with work the work of Von Neumann and Morgenstern in 1944 [64] and had arguably its most famous result in John Nash's landmark paper [63] published in 1951, which established the subject's must enduring result: every game where each decision-maker is restricted to have a finite number of choices has an appealing "solution" called a Nash equilibrium. The word "solution" is put in quotations because the idea of what constitutes a satisfactory solution to a game is itself a matter of research and debate. Alternate ways of looking at how a game is structured and played lead to different "solution concepts", of which Nash equilibrium is an example.

Of much more recent concern, particularly since the advent of the Internet as a source of meaningful games with large numbers of players with rich sets of actions, is the computational considerations implicit in different solution concepts. Indeed, for a group of players to attain a Nash equilibrium they would be faced with a computational problem, which prompts an immediate question: is there an efficient way to solve this problem? This is in the realm of computational complexity and the context for most of the results contained within this thesis. The offshoot of game theory which concerns itself with such questions has come to be known as algorithmic game theory [66]. The contention of many researchers in this field

is that if a solution concept does not admit efficient computation, then this puts into doubt the validity of that concept. Nash equilibria are one such example generally believed to be "difficult"[1] to compute and hence puts into question whether players would be able to make decisions according to its predictions [67].

Motivated by a search for games with efficiently computable solution concepts, my thesis explores the computational complexity of computing solution concepts in sequential and simultaneous games which involve at some point in their analysis a central concern with integer points in polyhedra. One example is when the actions of some or all players are described by integer vectors bounded by sets of linear inequalities. Another is when the outcomes of the game are sufficiently described by integer "counts" of players who play certain actions. Concern with integer decision variables has been well-studied in discrete optimization, and the work found here explores extensions of that work to game theoretic settings.

A fundamental contribution of the this thesis is the provision of an efficient means of "counting" equilibria; that is, we provide algorithms which count the exact number of equilibria in a game. As opposed to problems in discrete optimization where there is little interest in describing the number of optimal solutions – as long as you can find one of them – in game theoretic problems, the ability to give an exact count of the number of equilibria has some appealing consequences. Indeed, when a game has multiple equilibria there is an inherent challenge in predicting the outcome: which equilibria will arise? Will the players to be able to simultaneously coordinate their actions to arrive at any one of the equilibria? Thus it is common in studies in applied game theory and operations management to focus on game situations where there is a unique equilibrium; in other words, an equilibrium count of one [14]. This is a theme we will return to throughout the thesis and motivates the use of one

[1]It was recently shown that the problem is complete for the complexity class PPAD, which, informally speaking, is the class of computational problems for finding fixed points. For details see [67].

our key methodological tools: rational generating functions.

The following two sections describe the two major settings considered in this thesis.

1.1 Computing pure Nash equilibria

One solution concept that has received a significant amount of attention is pure strategy Nash equilibria of simultaneous, single-period games. We start with a few basic definitions.

A simultaneous, single-period game consists of a set of players $I = \{1, \ldots, n\}$, each with an associated *action* (or *pure strategy*) set $A_1, \ldots, A_n$. We refer to the set $A = \prod_{i=1}^{n} A_i$ as the *action profiles* of the game and for each profile $\mathbf{a} = (a_1, \ldots, a_n)$ and player i we associate a *utility* $u_i(\mathbf{a})$. A *pure-strategy Nash equilibrium* (PSNE for short) is an action profile $\mathbf{a}$ which is stable; that is, no one player would strictly benefit from deviating in their action given the actions of the remaining players are unchanged. Formally, $\mathbf{a} = (a_1, \ldots, a_n) \in A$ is a PSNE if for each $i \in \{1, \ldots, n\} : u_i(\mathbf{a}) \geq u_i(\mathbf{a}_{-i}, a'_i)$ for all $a'_i \in A_i$, where $\mathbf{a}_{-i} = (a_1, \ldots a_{i-1}, a_{i+1}, \ldots, a_n)$. A related concept called *mixed-strategy Nash equilibria* allows for players to choose a probability distribution over their action set A_i as a strategy and the equilibrium condition is stated in terms of expected utilities.

In the last decade there has been much interest in analyzing the complexity of computing mixed-strategy Nash equilibria [15, 21] as well as a slightly more general concept called correlated equilibria [68]. The next two chapters of this thesis focus on the question of computing PSNE, which on the one hand is more simple to understand than mixed and correlated equilibria, and on the other hand, can be more challenging to compute. Indeed, unlike mixed-strategy Nash equilibria, which are guaranteed to exist for finite games [63], their pure-strategy counterparts may not exist. Despite this difficulty, in many important classes of games finding PSNE is of great interest. One reason is that the concept of PSNE fits better our intuition that to play a game we must choose once and for all our action.

The appeal to some random process to decide one's action and then building a notion of equilibrium or stability around these random processes has been questioned by some authors [13, 14].

There are several important questions regarding the computation of PSNEs, notably the question of existence, but also in finding a sample equilibrium, counting and enumerating *all* equilibria, or optimizing functions over the set of pure equilibria. Some of the latter problems are more useful in understanding the structure of the game but come at higher computational costs. Most interest in the literature has centered around the computational complexity of arguably the simplest of these problems: deciding if a PSNE exists.

The complexity of deciding if a PSNE exists largely depends on how the game is *represented*. *Normal form* is the most familiar method of representing games. Normal form games take as input an explicit representation of each utility value $u_i(\mathbf{a})$ specifying the game, leading to an input size of $n|A_1|\cdots|A_n|$ numbers. The existence of PSNE can be decided in polynomial time in this input size, by checking every pure strategy profile. However, the size of the normal form representation grows exponentially in the number of players. Indeed, if each player has m actions this would mean specifying nm^n parameters, a representation size that is exponential in the number of players and actions. This is problematic in practice, especially since many games of interest involve a large numbers of players. On the other hand, sometimes games have an exponential number of actions in the natural input size of a game. A well-studied example is network congestion games, where the action sets of each player correspond to sets of paths on a network [39]. A normal form representation would be exponential in size in this case.

The normal form is generally considered an unacceptable way of representing a game from a computational perspective, and fortunately most games of practical interest have structured utility functions that can be represented more *compactly*. Indeed, few games of interest are best played by an exhaustive search over all possibilities, and often in these settings

players understand their payoffs in a more condensed way. This intuition has lead to the development of a major thrust in the literature – searching for *compact game representations* that succinctly describe games of interest and yet still yield efficient algorithms for computing equilibria on smaller input sizes.

In the following two chapters we explore compactness in three directions. The first is in compactly representing utility functions as opposed to "look-up" tables as in normal form. This is the most commonly explored direction of compactness in the literature. The second is in compactly representing the set of actions of each player. Lastly, we explore ways in which the set of players themselves can be compactly represented. The latter two types of compactness have further implications in terms of yielding smaller representations of utilities, which are ultimately the objects we need to represent in order to specify a game.

The task of compactly representing utilities has largely proceeded in three major directions. The first seeks to exploit *independence* structure in utility functions, meaning that each player's utility function may not depend explicitly on the entire action profile but instead on actions of some small subset of the player. The second direction is in representing utilities in functional form, where each utility value $u_i(\mathbf{a})$ is polynomial-time computable given a specified action profile $\mathbf{a}$ instead of being represented explicitly in the input of the game. The third direction is symmetry, where we make assumptions about the similarity of players in terms of their actions and payoffs which can drastically reduce the size of the representation required. We summarize the general approach of the first two directions here and discuss symmetry in Section 1.3.1 and Chapter 3.

Regarding the first direction, perhaps the most influential compact representation that exploits independence is graphical games [50], which encode independencies in a graphical structure. In other models this independence depends on actions taken by each player, called *context-specific independence*. One example is congestion games [70] where the payoffs of a given player depends on how many other players are playing the same or related action and

is independent of those players whose actions are unrelated. Congestion games have been shown to always possess a PSNE [70]. Similar reasoning led to the exploration of a more general class of games which are also guaranteed to have PSNE called potential games [60]. A mature example of this framework are action-graph games, which are fully-expressive game representations that compactly represent context-specific independence. The representation is based on the action graph, a directed graph with actions as nodes [10]. In these settings numerous questions surrounding the computation of PSNE has been explored often with favorable results when certain parameters are fixed. For instance, in graphical games exhibiting bounded treewidth, polynomial time dynamic programming algorithms can be used to compute PSNE [41].

For the compact representations above, exploiting the structure in utility functions often results in utility functions with a smaller domain. These smaller utility functions are then assumed to be explicitly represented. However, in practice these utility functions often have even shorter descriptions; for instance, as combinations of well-known mathematical functions. Functional representation of utilities are likely the most common form of utility in models where actions spaces are continuous and thus is prevalent in game theory models in the economics literature such as Cournot competition and Hotelling location games. In the computer science literature, functional forms of utility have also been considered. For instance, in the case of local effect games, a particular type of linear utility functions yields existence of potential functions and PSNE [59]. Devanur and Kannan [34] compute market equilibria when utilities are represented as piecewise linear concave functions. Overall, however, few positive results are known regarding efficient computation when utility is expressed in functional form.

As for negative results on the complexity of computing equilibria with utilities in compact form, there are some very general hardness results for the cases where utilities are presented by Boolean circuits or Turing Machines. This has been explored in the following general

class of games defined in [73]:

Definition 1.1.1 (Circuit game)**.** A *circuit game* is a game with set of action profiles A and utility functions $\{u_i, i = 1, \ldots, n\}$ which are specified by integers $k_1, \ldots, k_n$ and boolean circuits $C_1, \ldots, C_n$ such that $A_i \subseteq \{0,1\}^{k_i}$ and $u_i(\mathbf{a}) = C_i(\mathbf{a})$ when $\mathbf{a} \in A_1 \times \cdots \times A_n$ and undefined otherwise.

Note that a circuit game can encode games where each player has exponentially many pure-strategies in a polynomial amount of space since here the input for each player is k_i and the action set A_i has size $O(2^{k_i})$. It can be shown (Theorem 6.1 in [73]) that deciding if a circuit game has a PSNE is Σ_2^P-complete, even when the number of players is fixed.[2] Moreover, when the number of actions is fixed it is still NP-complete to decide (Theorem 6.2 in [73]). Similar results were discovered by Àlvarez et. al. [2], for the case where utilities are represented as deterministic Turing machines. Considering complexity alternatively for a fixed number of players or a fixed number of actions (but not both, otherwise the game is trivial) is a common tactic in the literature on computing PSNE and the two studies in Chapters 2 and 3 follow this pattern.

A game representation is of *polynomial type* if the number of players and numbers of actions per player are bounded by a polynomial in the size of the representation [68]. Game representations of *exponential type* are those with exponentially many players or actions. Efficient computation in this setting presents important challenges. In this thesis we explore two classes of games that are of exponential type: integer programming games and symmetric games with piecewise linear utilities. The contributions contained in this thesis to this area of research are summarized in Section 1.3.1 below.

[2]The complexity class Σ_2^P is reviewed in Section 1.4.1 below

1.2 Bilevel optimization

The second half of this thesis concentrates on another game setup, a simple sequential game where there are two players – a leader and a follower. The leader acts first, and the follower responds accordingly upon observing the leader's action. Early analysis of this model originates with the German economist Heinrich von Stackelberg in 1934 [84]. The problem domain we consider is a generalization of this early work into the now established field of bilevel optimization, which is essentially the study of sequential decision making from the perspective of theoretical optimization. The focus of bilevel optimization is on algorithms and efficient computation of solutions to these games (see [18] for a survey and introduction to the problem domain and the numerous references therein). Bilevel optimization models have been used to model a large variety of applied problems, including toll setting on transportation networks [56], revenue management [20] and competitive location planning [40], among many others.

The following is a formulation of a general bilevel optimization problem:

$$\begin{aligned} \inf \quad & F(\mathbf{x}, \mathbf{y}) && (1.1)\\ \text{s.t.} \quad & \mathbf{y} \in Y && (1.2)\\ & G(\mathbf{x}, \mathbf{y}) \leq 0 && (1.3)\\ & \mathbf{x} \in \operatorname{argmin}\{\, f(\mathbf{x}, \mathbf{y}) : \mathbf{x} \in X, g(\mathbf{x}, \mathbf{y}) \leq 0 \,\} && (1.4) \end{aligned}$$

The leader controls the $\mathbf{y}$ variable and the follower decides $\mathbf{x}$. The leader seeks to optimize $F(\mathbf{x}, \mathbf{y})$ which is a function of both variables. The leader seeks to influence the follower's

choice of $\mathbf{x}$ through adjusting the constraints and objectives of the lower level problem:

$$\begin{aligned} \min \quad & f(\mathbf{x}, \mathbf{y}) && (1.5) \\ \text{s.t.} \quad & \mathbf{x} \in X && (1.6) \\ & g(\mathbf{x}, \mathbf{y}) \leq 0 && (1.7) \end{aligned}$$

which is a *parametric optimization problem* in $\mathbf{y}$. For a pair of decisions $(\mathbf{x}, \mathbf{y})$ to be feasible they must satisfy the "upper-level" constraints (1.3)-(1.4), the "lower-level" constraints (1.7)-(1.7), and the argmin condition (1.4). Such a pair is then known as "bilevel feasible".

The fundamental object of study in bilevel optimization is precisely the set of *bilevel feasible solutions*:

$$\mathcal{F} = \{(\mathbf{x}, \mathbf{y}) : G(\mathbf{x}, \mathbf{y}) \leq 0, \mathbf{x} \in \operatorname{argmin}\{\, f(\mathbf{x}, \mathbf{y}) : \mathbf{x} \in X, g(\mathbf{x}, \mathbf{y}) \leq 0 \,\}\}. \tag{1.8}$$

Much of the complication associated with bilevel optimization is captured in the properties of this set. First, $\mathcal{F}$ is nonconvex in general, and difficult to describe efficiently [18]. Second, under certain scenarios it may not even be closed, in which case an optimal solution may not even exist [83]. The question of how one can identify when the infimum exists, either algorithmically or structurally, is explored in special cases in Chapters 4 and 5.

There is an even more important philosophical difficulty to solving bilevel integer programs. Note that the formulation (1.2)-(1.4) has an ambiguity built into it. As written, the formulation makes the implicit assertion of what is known in the bilevel literature as the *optimistic assumption* [18]. It is possible for the lower level problem to have multiple optimal solutions for a given $\mathbf{y}$. The follower is indifferent amongst these alternate optima, but the leader may prefer certain optima over others, and yet can exert no more direct influence over the follower's choice. How should the follower decide which of the alternate optima to

choose? To eliminate this difficulty the *optimistic assumption* is made whereby we assume that whenever the follower has a choice of optimal $\mathbf{x}$ for a given $\mathbf{y}$, the follower chooses the $\mathbf{x}$ that best suits the leader. This is reflected in the formulation (1.9) itself: the leader is modeled as choosing $\mathbf{x}$ subject to the argmin set of the lower level problem. The optimistic assumption represents the most common "solution concept" in bilevel optimization and is the focus of the work in this thesis.

An alternate "solution concept" is known as the *pessimistic assumption* whereby we assume the follower chooses $\mathbf{x}$ from amongst his alternative optima so as to punish the leader as much as possible. Although also prevalent in the literature, it is much less common than the optimistic approach and is often more difficult to analyze [28].

The vast majority of the work in bilevel optimization concerns the setting where both $\mathbf{x}$ and $\mathbf{y}$ are continuous. The main tools in the analysis of such problems is parametric linear and nonlinear optimization which often relies on differentiability and continuity assumptions [28]. A significant stream of research has focused on the linear case, where all objectives and constraints are linear. This special case is called bilevel linear programming (BLP). Although arguably one of the simplest general cases one might consider, it turns out that solving BLP is considered to be extremely difficult. Indeed, BLP is NP-hard to approximate to within a constant multiplicative factor [33].

The work in this thesis concerns an emerging trend in the bilevel optimization literature concerning the case where some or all of the decision variables are *integer*. We call this area *bilevel integer programming* (BILP). This direction has been heralded as an important direction in the maturation of the field [18] but also presents numerous challenges to the standard tools of analysis. Firstly, tools which involve differentiation do not apply directly to this setting and there are significant challenges even assuring that bilevel optimal solutions exist [83].

Much of the interest in this area lies in the fact that many applied settings involve

binary and integer decisions. Some applications considered to date include gas shipping [29], network interdiction [32], removal of hazardous materials [38] and product introduction [76]. Due to potential for applications, bilevel integer programming has been a topic of interest in the literature over the past two decades [5, 27, 32, 43, 46, 61, 72, 83, 85]. The contributions contained in this thesis to this area of research are summarized in Section 1.3.1 below.

1.3 Overview and summary of contributions

We can summarize the major contributions of this work into three areas. The first two concern results that are of interest in the respective fields of computing PSNE and solving bilevel optimization problems. The third area of contribution is the methodology whereby these results were derived, which we feel is of independent interest. Indeed, to our knowledge, these techniques are being used for the first time in their respective fields.

1.3.1 Contributions in the study of pure Nash equilibria

In this thesis we present efficient algorithms for computing PSNE for two classes of games of exponential type: integer programming games and symmetric games with piecewise linear utilities. We demonstrate how an explicit structure of utility, combined with novel methods used to encode equilibria, allow for polynomial-time algorithms for computing PSNE.

Integer programming games are simultaneous games where the players' actions are lattice points inside of polytopes described by systems of linear inequalities. Since the sets of actions are given implicitly by the description of polytopes, they may be of exponential size with respect to the input size. Thus, integer programming games are games of exponential type. It is straightforward to see that for an integer programming game with general polynomial-time computable utility functions the problem of deciding if a PSNE exists is Σ_2^P-hard (Proposition 2.2.2). In our setting, each player's payoffs are given a particular functional

form called *difference of piecewise linear convex (*DPLC*)* which can model a wide variety of utilities in functional form.

In our second setting we consider symmetric games. A game is *symmetric* when all players are identical and interchangeable. In a symmetric game, a player's utility depends only on his chosen action and the prevailing *configuration*; that is, the integer vector specifying the numbers of players choosing each of the actions. As a result, symmetric games can be represented more compactly than the normal form: we only need to specify a utility value for each action and each configuration. For a symmetric game with n players and m actions per player, the number of configurations is $\binom{n+m-1}{m-1}$. Thus we need only $m\binom{n+m-1}{m-1}$ numbers to specify the game.

Our point of departure in this chapter is in achieving further compactness of the representation of a symmetric game. We consider symmetric games with a functional representation of utilities that is much more compact than the standard explicit representation of utilities for each configuration. The compact form of utility we consider is again a natural one – piecewise linear functions of the configuration vector. Such piecewise linear functions can be represented by specifying a polytopal division of the space of configurations and an affine function for each polytope in the division. Our methods also allow for there to be an exponential number of players in the size of the input. We show that even when the number of actions is fixed, deciding the existence of equilibria in these games is NP-complete when utilities are represented as boolean circuits (Theorem 3.2.4).

In both cases the main results presented here are algorithms to count the number of pure Nash equilibria (thus deciding if such an equilibrium exists) and to find a sample pure Nash equilibria, when one exists. When certain parameters are fixed, these algorithms scale polynomially with size of representation, which in both cases includes the encoding of piecewise linear functions which define the utilities.

In the case of integer programming games, our algorithms run in polynomial time when

the number of *players*, dimensions of the polytopes that define the action sets, and the number of "convex" linear pieces in the payoffs are all fixed. In the case of symmetric games, our algorithms run in polynomial time when the number of *actions* is fixed and scale with the representation of the piecewise linear utilities. When this representation scales with $\log n$ we get exponential savings over existing algorithms [13].

One major difference between these two models is that one considers games with a fixed number of players and an exponential number of actions the other vice versa. As mentioned above, analyzing games in these two alternate cases – that of having a fixed number of players or a fixed number of actions – is a common approach in the literature and it is encouraging that our methods can say something significant in both cases. Indeed, all these results derive from our ability to express the set of pure equilibria, which is typically exponential in size, in a compact representation that has appealing computational properties – that of a rational generating function. This approach is the subject of Section 1.3.3 below. There are also important differences between the way the analysis unfolds in the two settings – integer programming games and symmetric games – which are discussed in Section 3.7.

We also further demonstrate, through a variety of applications, the computational advantages of encoding equilibria compactly as a generating function. For instance, we are able to efficiently compute equilibria that are Pareto optimal or maximize social welfare in the case of integer programming games. In the symmetric setting we are able to compute information about the distribution of social welfare over the set of equilibria, among other computations of interest.

One further application that demonstrates the power of our methods is in combining simultaneous and sequential decision-making. In the case of integer programming games we provide algorithms to compute Stackelberg–Nash equilibria for a sequential setting; that is, where there is a leader followed by multiple competing followers (see Section 2.5). In the case of symmetric games, we explore a setting where there is a mechanism designer who can

choose the number of players and the structure of their utility functions in order to achieve certain outcomes, such as maximizing the worst case payoff to a player.

1.3.2 Contributions to the study of bilevel optimization

The work in Chapters 4 and 5 concerns the search for efficient algorithms and useful structure in the study of bilevel integer programming. To be precise, we consider bilevel integer programs which fit into the general framework:

$$\begin{aligned} \inf \quad & f(\mathbf{x}, \mathbf{y}, \mathbf{z}) \\ \text{s.t.} \quad & (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in P \subseteq \mathbb{R}^{n+p+d} \\ & \mathbf{x} \in \operatorname{argmin}\{\, \psi(\mathbf{y}) \cdot \mathbf{x}' : A\mathbf{x}' \leq \pi(\mathbf{z}),\, \mathbf{x}' \in \mathbb{Z}^n \,\} \end{aligned} \tag{1.9}$$

where $A \in \mathbb{Z}^{m \times n}$, f is a real-valued function on domain P which is assumed to be bounded, and the functions $\psi : \mathbb{R}^p \to \mathbb{R}^n$ and $\pi : \mathbb{R}^d \to \mathbb{R}^m$ are vector-valued. As we explore different special cases of this model, further conditions and assumptions will be stated.

The special case of (1.9) considered most in the literature is that of fixed-unit-cost bilevel integer programs (i.e., (1.9) when y is fixed) and where f is linear, P is assumed to be a bounded polyhedron and the functions $\psi : \mathbb{R}^p \to \mathbb{R}^n$ and $\pi : \mathbb{R}^d \to \mathbb{R}^m$ are affine. We call this setting *bilevel mixed integer linear programming* or BMILP for short, reflecting the possibility that either $\mathbf{y}$ or $\mathbf{z}$ are continuous. Existing approaches to solving this problem use standard ideas from integer optimization, such as branch-and-bound enumeration [5] and cutting planes [32]. But, as is well documented, there are numerous pitfalls when applying standard ideas of branch-and-bound or cutting plane methods to a bilevel setting (see [32] for a discussion). For instance, one pitfall is that linear relaxations may have integer solutions that are not feasible to the original problem. To date, no algorithms have been proposed that deal with the discrete nature of the problem directly rather than recourse to

linear relaxations. Moreover, all known algorithms are based on enumeration and branching and have exponential running times in the worst case. Despite some promise for practical implementation of these algorithms (see [32]) no algorithms with a provably better running time have been proposed. This is remedied in Chapter 4 of this thesis, where we describe polynomial time algorithms in fixed dimension for solving BMILPs.

We apply recent results from the study of parametric integer programming, based on the work of [37] and [49] to yield algorithms to solve BMILPs that run in polynomial time when $\mathbf{x}$ is integer, $\mathbf{z}$ is continuous and the dimension n is fixed. Implicit in our approach is an algorithm to decide if a BMILP has any optimal solutions at all, acknowledging the possibility of an open bilevel feasible set $\mathcal{F}$, something lacking in the literature.

Using generating function methods we also derive results for counting and enumerating alternate optima when $\mathbf{z}$ is discrete. Lastly, we explore two extensions of this basic model. The first is to a special case of nonlinear discrete bilevel optimization where the objective f is polynomial. The second generalization is to a multi-objective bilevel integer programming problem where the follower has r linear objectives representing competing interests. In both cases we are able to derive polynomial time algorithms when appropriate dimensions are fixed.

Chapter 5 concerns (1.9) in its generality. The current literature has almost exclusively explored special cases of (1.9) where either the leader's decision affects the follower's unit cost or right-hand side but not both (that is, one of either $\mathbf{y}$ or $\mathbf{z}$ is fixed, see [27] for the latter case). To our knowledge, simultaneous changes in the unit cost and right-hand side of the lower level problem have not been studied systematically. The only known algorithm which applies to this more general setting is an algorithm for nonlinear discrete bilevel optimization problems which uses a transformation to complicated continuous global optimization problems [43] and only provides algorithms to find approximate solutions.

Chapter 5 explores structural and existence results for this variety of bilevel integer

programs. The main result is a decomposition of the original problem into a finite set of subproblems with fixed-unit cost linear lower level objectives. This is achieved by identifying a finite number of equivalence classes for the leader's choice of $\mathbf{y}$; which turn out to be the relative interiors of cells in a polyhedral fan. We explore various aspects of this decomposition including the number of subproblems generated by the decomposition and how to characterize each equivalence class. Using the decomposition we also present algorithms to a special case where all variables are integer, the constraints are linear and leader's objective functions is either linear or a nonnegative polynomial; in which case each subproblem can be solved in polynomial time in fixed dimension. In addition, we explore the existence of optimal solutions when the leader's decision variables are continuous and provide sufficient conditions for existence in special settings.

1.3.3 Analytical approach

An important aspect of the novelty of the results presented in this thesis are the analytical tools used to derive them. In particular, this work draws upon two developments in the theory of discrete optimization that occurred during the 1990's and applies them to game theoretic settings. The first development is the efficient encoding of lattice point sets by generating functions discovered in [6] and significantly extended in [8]. This development provides the main tool of analysis in Chapters 2 and 3 and also play an important role in the remaining chapters. The second development is the application of Gröbner basis methods to the study of integer programming initiated by [19]. It plays a significant role in Chapter 5, providing interesting geometric structure to an otherwise challenging setting.

These developments initiated streams of research in exploring the theoretical and practical implications of these methods to discrete optimization as well as other areas of applied mathematics. The literature surrounding Gröbner basis methods focused mainly on connections between Gröbner bases and other geometric and algebraic structures (see [78] and

[81] for example), and less on applying these methods to practical optimization problems. One of the major reasons for this historical evolution is the fact that general Gröbner basis computations have inefficient running times, both in theory and practice.

In contrast, the literature developed around Barvinok's generating functions found more success in assuring good theoretical running times when used to solve integer programs, and initiated computer implementations [26, 51, 82] and numerical studies [22].

Nonetheless, to this point, except for some special cases, neither method has been able to compete with the computational strength of standard algorithms for solving integer linear programs, such as branch-and-bound and cutting planes, and their numerous powerful implementations. Of the many reasons for why this is the case, we focus on two that are most relevant in terms of motivating the work in this thesis.

Firstly, Gröbner basis methods provide algebraic structures that in some sense contain "too much" information when one is only interested in solving a single integer program. Indeed, if Gröbner basis methods are used to solve the program $\min\{\mathbf{c} \cdot \mathbf{x} : A\mathbf{x} = \mathbf{b},\ \mathbf{x} \in \mathbb{N}^n\}$, then the majority of work has also been done for solving the related integer programs $\min\{\mathbf{c}\cdot\mathbf{x} : A\mathbf{x} = \mathbf{b}',\ \mathbf{x} \in \mathbb{N}^n\}$ for any $\mathbf{b}'$. That is, Gröbner basis methods provide "parametric" information about an integer program is superfluous for basic calculation.

Secondly, generating functions also contain, in some sense, more information than is needed to solve a single integer program. Indeed, using them we can encode *all* lattice points inside a polytope, which is a significant amount of information and can be used to answer numerous other questions besides integer linear optimization; including approximate integer polynomial optimization [24] and enumeration of lattice points [23]. This thesis further demonstrates that generating functions of lattice point sets can be combined to characterize optimality or equilibrium conditions in more complicated settings than integer linear programming.

So, despite a lack of computational efficiency when solving general integer linear programs,

what is clear is that the algebraic constructs used in both methods contain information that gives insight into the structure of solutions. Some attention has been given to exploring situations where this structural information can be used to good advantage and where traditional methods often fail to provide "enough" information or structure to yield convincing algorithms. One direction is in exploring special cases, such as "N-fold integer programs" [25]. Another is in considering more elaborate settings, such as the recent application of generating function methods to multicriteria discrete optimization [23].

This thesis continues these explorations in the area of game theory and bilevel optimization. It is hoped that researchers in the fields of game theory and bilevel optimization can gain an appreciation of these "nonstandard" tools, and those studying generating functions and Gröbner basis methods find other avenues wherein to apply their techniques.

The key idea behind our analysis is that we view each problem as having embedded parametric integer programming problems and lattice point enumeration problems and then we take advantage of the structure and algorithms afforded by Gröbner bases and generating functions. In the case of generating functions our perspective is to view sets of equilibria as lattice point sets inside unions, set differences and intersections of polyhedra. Finding equilibria thus reduces to understanding the structure of these lattice point sets. The complexity results explored throughout the thesis derive directly from results for generating function algorithms in fixed dimension. By these novel tools of analysis we are able to provide new results not readily accessible to existing methods in the respective literatures on game theory and bilevel optimization.

1.4 Some background topics

The last section of this introductory chapter contains a brief introduction to two background topics that are used in the work that follows. The first is a brief review of the complexity

class Σ_2^P, which has already been referred to above. The second is a brief introduction to the theory of generating functions, which we will have occasion to refer to in each of the remaining chapters of the thesis. Gröbner basis methods only appear in the final chapter and are thus introduced there.

1.4.1 The complexity class Σ_2^P

The major goal of this section is to establish some intuition for the complexity class Σ_2^P, which is the natural complexity class for the problem of finding pure Nash equilibria in games of exponential type. A more detailed description of this complexity class and its generalization to the so-called "polynomial hierarchy" can be found in the article which first introduced the concept [77]. This presentation is based on Chapter 5 of the introductory text on complexity theory by Arora and Barak [4] and assumes familiarity with some basic theory of computation including deterministic Turing machines and describing decision problems as languages on a finite alphabet (see Chapter 1 of [4] for an introduction to these concepts).

The class Σ_2^P is a generalization of P, NP and $co-\text{NP}$ defined as follows:

Definition 1.4.1. The class Σ_2^P is the set of languages L for which there exists a polynomial time deterministic Turing machine T and a polynomial p such that

$$I \in L \iff \exists u \in \{0,1\}^{p(|I|)},\ \forall v \in \{0,1\}^{p(|I|)},\ T(I,u,v) = 1 \tag{1.10}$$

where $|I|$ is the binary encoding size of instance I.

There are three alternate definitions of Σ_2^P, see Chapters 5 and 6 of [4] for details. To help solidify the given definition we show the following decision problem is in Σ_2^P.

Example 1.4.2. Consider the language:

$$\textsc{MaxClique} = \{\langle G, k\rangle :\ \text{the largest clique in graph } G \text{ has size exactly } k\}.$$

We first observe that there seems to be no short certificate for an instance $\langle G, k\rangle$ to belong to MaxClique, since even if you demonstrate that there is a clique in G is size k it is difficult to show that *every* other clique is smaller than this size.

However, it is straightforward to see that MaxClique is in Σ_2^P, since it is apparent that instance $\langle G, k\rangle$ is in MaxClique if and only if

$$\exists S \; \forall S' \; S \text{ is a clique of size } k \text{ and } S' \text{ is not a clique of size} \geq k+1 \tag{1.11}$$

Testing whether a pairs of subsets of (S, S') satisfy this condition in the graph G can be done in polynomial time (thus defining the polynomial-time Turing machine in Definition 1.4.1) and clearly the S and S' are polynomial in encoding size.

1.4.2 Rational generating functions

We here provide a brief and selective overview of the theory of rational generating functions. We have two aims: to introduce some machinery that is used in proving our main theorems in subsequent chapters, and to invite other researchers to use these methods in future work.

Generating functions offer the key benefit of compactly representing exponential-cardinality sets of integer points while efficiently supporting the computational operations of counting and enumerating points in the set. We demonstrate the essence of the approach in the following simple example. Consider the set of integers on the line between 0 and n. We can represent these points by associating an exponent of a complex variable ξ with every integer $x \in [0, n]$. The choice of variable ξ plays an important role in later analysis, where residue calculus can be used to extract important information.

Using this encoding we can represent the integers in the interval $[0, n]$ as the exponents in the polynomial expression

$$\sum_{x=0}^{n} \xi^x, \tag{1.12}$$

called a *generating function representation*. So far we have not gained much: there are exponentially many terms in (1.12) (in terms of the binary encoding size of n), just as there are in an explicit listing of the numbers $0, 1, \ldots, n$. However, we can also write the expression as

$$\sum_{x=0}^{n} \xi^x = \frac{1}{1-\xi} - \frac{\xi^{n+1}}{1-\xi}. \tag{1.13}$$

Thus, we have written the *long* sum (1.12) as a *short* sum of two rational functions, called a *rational generating function representation*. The encoding length of this new representation is now *polynomial* in the encoding length of n. Not only does this representation appeal because of its compact size, but it also offers computational benefits. Observe that we can compute the cardinality of our set by setting $\xi = 1$ and then evaluating the sum. Working with Equation (1.12), this requires an exponential number of arithmetic operations. However, we can obtain the same result by working with (1.13) (letting $\xi \to 1$ and using L'Hôpital's rule), thereby performing exponentially-fewer arithmetic operations.

We are often interested in sets arising from unions, intersections and differences of simpler sets of integers. A basic illustration is as follows: suppose we have the set $\{0, \ldots, n\}$ represented by the rational generating function $\frac{1-\xi^{n+1}}{1-\xi}$, as well as the set $\{n+1, \ldots, 2n\}$ represented by the rational generating function $\frac{\xi^{n+1}-\xi^{2n+1}}{1-\xi}$. A rational generating function representing the *union* of these two *disjoint* sets can by found by *summing* the representations of $\{0, \ldots, n\}$ and $\{n+1, \ldots, 2n\}$:

$$\frac{1-\xi^{n+1}}{1-\xi} + \frac{\xi^{n+1}-\xi^{2n+1}}{1-\xi} = \frac{1-\xi^{2n+1}}{1-\xi}.$$

It is straightforward to verify that $\frac{1-\xi^{2n+1}}{1-\xi}$ is a rational generating function encoding of the union $\{0, \ldots, 2n\}$. A more general result obtains when combining sets that are not disjoint (Lemma 1.4.4 below). Observe that the complication in this case is that we cannot simply sum the two generating function representations, as this will yield non-unitary weights on

the terms in the new generating function that correspond to points lying in the intersection of the two sets.

Working with rational generating functions is of course unnecessary in the simple setting we have discussed so far, but it becomes useful when extended to deal with more general sets of integer points in higher dimensions. A key milestone was the work of Barvinok [6], who introduced a polynomial-time algorithm for representing as a rational generating function the integer points inside of a *rational polytope* $P \subseteq \mathbb{R}^m$ given by an inequality system $\{\mathbf{x} \in \mathbb{R}^m : M\mathbf{x} \leq \mathbf{b}\}$, provided the dimension m is fixed. Note that representing the integer set $P \cap \mathbb{Z}^m$ by the inequality description of P gives little hint as to its cardinality. However, as in our simple example, we shall see that a generating function representation allows us to count the number of integer points in P exactly in polynomial time.

Now we briefly describe Barvinok's algorithm and its useful extensions and applications. Consider the *generating function* of the lattice point set $P \cap \mathbb{Z}^m$, which is defined as

$$\begin{aligned} g(P \cap \mathbb{Z}^m; \boldsymbol{\xi}) &= \sum_{\mathbf{x} \in P \cap \mathbb{Z}^m} \boldsymbol{\xi}^{\mathbf{x}} \\ &= \sum_{\mathbf{x} \in P \cap \mathbb{Z}^m} \xi_1^{x_1} \cdots \xi_m^{x_m}. \end{aligned} \tag{1.14}$$

Note that each lattice point $\mathbf{x}$ in P is mapped to the exponent of a monomial $\boldsymbol{\xi}^{\mathbf{x}}$ in $g(P \cap \mathbb{Z}^m; \boldsymbol{\xi})$.

Lemma 1.4.3 (**Barvinok's Theorem** [6])**.** *Let P be a polytope in $\mathbb{R}^m$ and $S = P \cap \mathbb{Z}^m$ with generating function $g(S, \boldsymbol{\xi})$ given by* (1.14) *which encodes the lattice points inside P. Then, there exists an algorithm which computes an equivalent representation of the form*

$$g(S; \boldsymbol{\xi}) = \sum_{i \in I} \gamma_i \frac{\boldsymbol{\xi}^{\mathbf{c}_i}}{\prod_{k=1}^{m} (1 - \boldsymbol{\xi}^{\mathbf{d}_{ik}})}, \tag{1.15}$$

where I is a polynomial-size index set, $\mathbf{c}_i, \mathbf{d}_{ik} \in \mathbb{Z}^m$ for all i and k. A formula of this type

is called a short rational generating function*. The algorithm runs in* polynomial time *when the dimension m is fixed.*

Note that the number of binomial terms in the denominator of each rational term is m and thus fixed when the dimension is fixed. When a lattice point set S is expressed in the form of (1.15) we refer to $g(S;\boldsymbol{\xi})$ as its *Barvinok encoding*. In the algorithms that follow, when a set S is given as input or output by its Barvinok encoding $g(S,\boldsymbol{\xi})$, the encoding size is the binary encoding of the integer vectors $\gamma, \mathbf{c}_i, \mathbf{d}_{i1}, \ldots, \mathbf{d}_{im}$ for $i \in I$.

It is important to note that Lemma 1.4.3 only describes how to encode sets of integer points inside of polytopes. The key result that makes this theory useful in our setting is that more general lattice point sets arising from simple operations on polytopal lattice point sets also admit Barvinok encodings. Barvinok and Woods [8] developed powerful algorithms that apply to these more general settings. For our purposes the most important algorithm concerns constant-length Boolean combinations of polytopes. A *Boolean combination* of the sets $S_1, \ldots, S_k$ is any combination of unions, intersections and set differences of those sets. For instance, $(S_1 \cap S_2) \setminus S_3$ is a Boolean combination of the sets S_1, S_2 and S_3.

Lemma 1.4.4 (**Boolean Operations Lemma** [8, Cor. 3.7])**.** *Given fixed integers k and ℓ there exists a constant s and a polynomial-time algorithm for the following problem. Given as* input*, in binary encoding,*

(I_1) *the dimension m and*

(I_2) *Barvinok encodings of k finite sets $S_p \subseteq \mathbb{Z}^m$, $g(S_p;\boldsymbol{\xi})$ such that for each rational term the number of binomials in the denominator is at most ℓ,*

output*, in binary encoding,*

(O_1) *rational numbers γ_i, integer vectors $\mathbf{c}_i$, $\mathbf{d}_{ij}$ for $i \in I$, $j = 1, \ldots, s_i$, where $s_i \leq s$,*

such that

$$g(S;\boldsymbol{\xi}) = \sum_{i\in I} \gamma_i \frac{\boldsymbol{\xi}^{\mathbf{c}_i}}{\prod_{k=1}^{m}(1-\boldsymbol{\xi}^{\mathbf{d}_{ik}})}$$

is a rational generating function of the finite set S that is the Boolean combination *of the sets $S_1, \ldots, S_k$, and where each rational term in the expression has at most s terms in its denominator and where I is a polynomial-sized index set.*

We remark that if k were allowed to vary, the number of binomials in the denominators (which essentially double at with each operation) would become exponential; thus, we must require that k be fixed in order to achieve a polynomial run time. We also note that if the input sets $S_p \subseteq \mathbb{Z}^m$ are integer points inside of polyhedra whose Barvinok encodings $g(S;\boldsymbol{\xi})$ arise from applying Lemma 1.4.3 then the condition that the number of binomials ℓ in the denominators are fixed follows under the assumption that the dimension m is fixed.

The essential part of the construction of Lemma 1.4.4 is the implementation of set intersections, which are based on the *Hadamard product* [8, Definition 3.2], a bilinear extension of the operation defined on monomials as

$$\alpha\mathbf{x}^{\boldsymbol{\xi}} * \alpha'\mathbf{x}^{\boldsymbol{\xi}'} = \begin{cases} \alpha\alpha'\mathbf{x}^{\boldsymbol{\xi}} & \text{if } \boldsymbol{\xi} = \boldsymbol{\xi}', \\ 0 & \text{otherwise.} \end{cases}$$

With this definition, clearly

$$g(S_1 \cap S_2) = g(S_1;\boldsymbol{\xi}) * g(S_2;\boldsymbol{\xi}).$$

Hadamard products can be computed in polynomial time on short rational generating functions [8].

Disjoint unions are a special case of combining sets. Thus, we could address them as well using the Boolean Operations Lemma. However, in this case there is an obvious stronger

result, analogous to our method in the simple example above. This allows the number of sets k in the union to be polynomial in the input size instead of fixed.

Lemma 1.4.5 (**Disjoint Unions**). *If two lattice point sets S and T are disjoint then the generating function for $S \cup T$ is the sum of generating functions for S and T. More generally, for disjoint lattice point sets $S_1, \ldots, S_k$:*

$$g\left(\biguplus_{i=1}^{k} S_i, \boldsymbol{\xi}\right) = \sum_{i=1}^{k} g(S_i, \boldsymbol{\xi}),$$

where $\biguplus$ denotes disjoint union.

Another powerful method to define lattice point sets is by integer projections. Let $S \subseteq \mathbb{Z}^m$ be a finite lattice point set, given by its rational generating function $g(S; \boldsymbol{\xi})$. Let $\psi \colon \mathbb{Z}^m \to \mathbb{Z}^d$ be a linear function and denote by $T = \psi(S)$ the image (projection) of S.

Remark 1.4.6. If the map ψ is one-to-one (injective) from S, then the generating function $g(T; \boldsymbol{\eta})$ of the projection T can be computed by making a *monomial substitution* in $g(S; \boldsymbol{\xi})$; see [8, Theorem 2.6]. This fact is used in the proof of Corollary 2.3.8.

When S is the set of lattice points in a polytope P, the following integer projection method of [8] can be employed to construct a rational generating function of the projection T, even when the mapping ψ is not one-to-one.

Lemma 1.4.7 (**Projection Lemma** [8, Theorem 1.7]). *Let the dimension m be a fixed constant. Then there exists a constant $s = s(m)$ and a polynomial-time algorithm for the following problem. Given as* input*, in binary encoding,*

(I_1) *an inequality description of a rational polytope $P \subset \mathbb{R}^m$;*

(I_2) *a positive integer d; and*

(I_3) *a linear map $\psi \colon \mathbb{Z}^m \to \mathbb{Z}^d$ given by an integral matrix;*

output*, in binary encoding,*

(O_1) *rational numbers* γ_i*, integer vectors* $\mathbf{c}_i$*,* $\mathbf{d}_{ij}$ *for* $i \in I$*,* $j = 1, \ldots, s_i$*, where* $s_i \leq s$*, such that*

$$g(T; \boldsymbol{\xi}) = \sum_{i \in I} \gamma_i \frac{\boldsymbol{\xi}^{\mathbf{c}_i}}{\prod_{k=1}^{m} (1 - \boldsymbol{\xi}^{\mathbf{d}_{ik}})}$$

is a rational generating function of the set $T = \psi(P \cap \mathbb{Z}^m)$*.*

Once a rational generating function of a set S has been computed, various pieces of information can be extracted from it. First, we consider computing the cardinality of S.

Lemma 1.4.8 (**Counting Lemma** [6])**.** *Let the dimension* m *be a fixed constant. Given a lattice point set* $S \in \mathbb{Z}^m$ *input as its Barvinok encoding* $g(S, \boldsymbol{\xi})$*, there exists a polynomial-time algorithm for computing* $|S|$*.*

The idea behind the proof of this result is analogous to the basic example at the beginning of this section. Given a Barvinok encoding of a lattice point set S as in (1.15), each of the basic rational functions has poles (the point $\boldsymbol{\xi} = \mathbf{1}$ in particular is a pole of all the basic rational functions), but after summing up only removable singularities remain. Obtaining the exact number of lattice points of lattice point set S is easy in (1.14), since clearly $|S| = g(S; \mathbf{1})$. Since (1.15) is a formula for the same function (except for removable singularities), we also have $|S| = \lim_{\boldsymbol{\xi} \to \mathbf{1}} g(S; \boldsymbol{\xi})$, which can be evaluated in polynomial time by performing a residue calculation with each basic rational function in the sum (1.15).

We can also *explicitly enumerate* all elements of S. We note that the cardinality of S can be exponential in the encoding length. Nevertheless there exists a *polynomial-space, polynomial-delay enumeration algorithm.* Such an algorithm only uses space that is polynomial in the encoding length of the input data. In addition, the time spent between outputting two items, and before outputting the first item and after outputting the last item, is bounded by a polynomial in the encoding length of the input data.

Lemma 1.4.9 (**Enumeration Lemma** [23, Theorem 7])**.** *Let the dimension m and the maximum number ℓ of binomials in the denominators be fixed. Then there exists a polynomial-space, polynomial-delay enumeration algorithm for the following enumeration problem. Given as* input*, in binary encoding, an integer bound $M \in \mathbb{Z}$ and the Barvinok encoding $g(S, \boldsymbol{\xi})$ of a lattice point set $S \in [-M, M]^m \cap \mathbb{Z}^m$,* output*, in binary encoding, all points in S in lexicographic order.*

The algorithm relies on a binary search procedure that uses cardinality counts to test for emptiness of each search region. Binary search can also give useful results for *optimizing*. By applying Lemma 1.4.4 and a binary search argument, we can optimize in polynomial time any linear objective over a set of lattice points in Barvinok encoding:

Lemma 1.4.10 (**Linear Optimization Lemma** [6])**.** *Let the dimension m and the maximum number ℓ of binomials in the denominator be fixed. Then there exists a polynomial-time algorithm for the following problem. Given as* input*, in binary encoding, an integer bound $M \in \mathbb{Z}$, integer vector $\mathbf{c} \in \mathbb{Z}^m$ and the Barvinok encoding $g(S, \boldsymbol{\xi})$ of a lattice point set $S \in [-M, M]^m \cap \mathbb{Z}^m$,* output*, in binary encoding, an optimal solution $\mathbf{x}^* \in \mathbb{Z}^m$ of the optimization problem* $\max\{\, \mathbf{c} \cdot \mathbf{x} : \mathbf{x} \in S \,\}$.

When the objective function is an arbitrary polynomial function (without any assumptions on convexity) that is nonnegative on a set with a Barvinok encoding, then it is still possible to use a fully polynomial time approximation scheme (FPTAS):

Lemma 1.4.11 (**FPTAS for maximizing non-negative polynomials** [23, Theorem 14])**.** *Let the dimension m and the maximum number ℓ of binomials in the denominator be fixed. There exists a polynomial-time algorithm for the following problem. Given as* input

(I_1) *two vectors $\mathbf{x}_\mathrm{L}$, $\mathbf{x}_\mathrm{U} \in \mathbb{Z}^m$,*

(I_2) *a Barvinok encoding of a finite set $S \subseteq \mathbb{Z}^m$ of lattice points that is contained in the box $\{\, \mathbf{x} : \mathbf{x}_\mathrm{L} \leq \mathbf{x} \leq \mathbf{x}_\mathrm{U} \,\}$,*

(I_3) *a list of coefficients $f_i \in \mathbb{Q}$, encoded in binary encoding, and exponent vectors $\boldsymbol{\alpha}_i \in \mathbb{Z}_+$, encoded in unary encoding, representing a polynomial*

$$f = \sum_i f_i \mathbf{x}^{\boldsymbol{\alpha}_i}$$

that is non-negative on S,

(I_4) *a positive rational number $1/\epsilon$ encoded in unary encoding,*

output*, in binary encoding,*

(O_1) *a point $\mathbf{x}_\epsilon \in S$ that satisfies*

$$f(\mathbf{x}_\epsilon) \geq (1-\epsilon) f^* \quad \textit{where} \quad f^* = \max_{\mathbf{x} \in S} f(\mathbf{x}).$$

An ingredient in the proof of the FPTAS which will be used in other contexts is the following result due to Barvinok [7].

Lemma 1.4.12. *Let the dimension m be fixed. There exists a polynomial-time algorithm for the following problem. Given as* input

(I_1) *a Barvinok encoding of a finite set $S \subseteq \mathbb{Z}^m$ of lattice points*

(I_2) *a list of coefficients $f_i \in \mathbb{Q}$, encoded in binary encoding, and exponent vectors $\boldsymbol{\alpha}_i \in \mathbb{Z}_+$, encoded in unary encoding, representing a polynomial*

$$f = \sum_i f_i \mathbf{x}^{\boldsymbol{\alpha}_i}$$

that is non-negative on S,

output*, in binary encoding,*

(O_1) *the sum* $\sum_{\mathbf{x}\in S} f(\mathbf{x})$

Here we have offered only some highlights from the theory of rational generating functions, focusing on results needed for the analysis in this paper. A more complete picture of this theory can be obtained from the excellent textbook by Beck and Robbins [9].

Remark 1.4.13. We end this chapter with a few remarks on the need for caution when considering the practical implications of complexity results using the above-stated theorems. Although the algorithms presented have nice theoretical run-times, in practice the situation in less promising. Many of these algorithms have been implemented in two software packages: `LattE machiatto` [51] and `barvinok` [82]. The algorithm of Theorem 1.4.3 for encoding polyhedra is quite sensitive to dimension and problems with up to dimension 30 are the largest instances that have been computed in practice [52]. The algorithms for boolean operations also requires costly operations and can deal with instances with dimension only up to 15. Potentially more disconcerting is that the only working implementation of the Projection Lemma has performed well on problems with only very few variables (up to four) [53]. Appeal to the Projection Lemma is considered most troubling from a practical point of view, and algorithms which avoid it entirely are preferred, although this is not always possible. This is an active area of research and it is hoped that future gains can be made in the practical implementations of these algorithms. Indeed, part of the motivation for improved implementations is the potential for application to some of the problems contained in this thesis [53].

Chapter 2

Rational generating functions and integer programming games

2.1 Introduction

In this chapter we introduce a new representation of strategic games called *integer programming games*, which are simultaneous games where the players' actions (pure strategies) are lattice points (i.e., integer points) inside polytopes described by systems of linear inequalities. Integer programming games are games of exponential type: the number of actions for each player is exponential in the input size. The number of actions corresponds to the number of lattice points inside a compactly-described polytope, which can be exponential in number even when the dimension of the polytope is fixed. One motivation for studying integer programming games is to explore the extent to which we can leverage the tools of integer programming, which have shown their undoubted value in modeling situations where there is a single decision-maker, to the multiple decision-maker setting. This paper describes one methodology – Barvinok's theory of short rational generating functions – that has been used for standard integer programming and how it can fruitfully extend to game scenarios.

It is straightforward to see that for an integer programming game with general polynomial-time computable utility functions, the problem of deciding if a PSNE exists is Σ_2^P-hard via a reduction from circuit games (Proposition 2.2.2). Our contribution is to explore a specific type of functional form of utility where each player's utility is given as a piecewise linear

function expressed in *difference of piecewise linear convex (*DPLC*)* form. Integer programming games with DPLC payoffs can express general normal form games (Proposition 2.2.4), but complexity gains from this representation are had only when the number of players and the polytopes defining the actions lie in a fixed-dimensional space. An example of a game that takes advantage of these complexity gains is given in Example 2.2.5.

The main result of the chapter is that the set of PSNE of integer programming games with DPLC payoffs can be encoded in a polynomial-sized data structure – a rational generating function – which admits efficient computation (Theorem 2.3.7). This encoding yields as corollaries polynomial time algorithms for counting and finding PSNE when the number of players is fixed and the number of pieces in the piecewise linear payoffs is restricted (Corollaries 2.3.9 and 2.3.10). The input is an inequality description of the polytopes defining the action sets and the integer vectors which define the affine pieces of the DPLC utility functions.

The theoretical gain here is in the fact that even although the number of actions is exponential in the input size, we can avoid explicit description of payoffs using Barvinok's theory of rational generating functions to "check" exponentially many possible deviations in polynomial time. A short rational generating function encoding is a convenient data structure for answering other questions regarding the structure of PSNE and other related concepts.

The main contributions of the chapter are:

- Introducing the use of Barvinok's theory of short rational generating functions to the study of strategic games and demonstrating the power of encoding sets of PSNE as generating functions.

- Presenting a tractable class of games, integer programming games with DPLC payoffs, for which PSNE and related quantities can be computed in polynomial time when

certain dimensions are fixed.

Also of note are two ideas used in several places in this chapter:

- In order to represent sets of equilibria, or other sets of interest, as rational generating functions we express the set as an overall feasible set in which unwanted elements, expressed as the union of projections of lattice point sets in polytopes, are removed. See for instance the proof of Theorem 2.3.7 where the set of PSNE is defined in this way. This is a general technique that is adapted to various settings in Sections 2.3 to 2.5.

- Some results are easier to show when the actions for each player in the game are *extended* to include a component that captures the payoff of that player. This extension allows for descriptions of *extended* strategy profiles and equilibria that are more amenable to generating function techniques and can readily be converted back to normal strategy profiles and equilibria. See for instance Definition 2.3.3.

Both of these ideas appear again in Chapter 3, but in a different context.

This chapter is organized into the following sections. Section 2.2 introduces integer programming games and discusses an application of this framework to a model of competing firms producing identical indivisible goods. Section 2.3 contains the main result of the chapter – demonstrating how generating functions can be used to encode sets of PSNE. Section 2.4 details several other applications of generating function constructions to the computation of Pareto optima, the price of anarchy, and pure minmax values. Lastly, Section 2.5 describes a sequential (Stackelberg–Nash) version of an integer programming game where a leader's actions affects the description of the polytopes defining the actions sets of a group of followers, who then play a simultaneous game.

2.2 Integer programming games

We begin by introducing the following class of strategic games:

Definition 2.2.1 (**Integer Programming Game**)**.** An *integer programming game* with n players is a noncooperative game where the set S_i of actions for each player $i \in I = \{1, \ldots, n\}$ is a set of lattice points inside a polytope; that is,

$$S_i = P_i \cap \mathbb{Z}^{d_i} \tag{2.1}$$

where $P_i = \{\mathbf{x} \in \mathbb{R}^{d_i} : M_i\mathbf{x} \leq \mathbf{b}_i\}$ is a rational polytope.

The set S of action profiles $\mathbf{s} = (\mathbf{s}_1, \ldots, \mathbf{s}_n)$ is the Cartesian product of the S_i's: $S = \prod_{i=1}^{n} S_i \subseteq \mathbb{Z}^d$ where $d = d_1 + \cdots + d_n$. The payoff functions are integer-valued of the form $u_i : S \to \mathbb{Z}$ for $i \in I$.[3]

As noted in the introduction, a distinguishing feature of this class of games is that the action sets S_i are defined succinctly by linear systems $M_i\mathbf{x} \leq \mathbf{b}_i$, even though $|S_i|$ may be exponential in size with respect to the input size of M_i and $\mathbf{b}_i$. We use rational generating functions to avoid explicitly enumerating each player's action set.

Now to the definition of payoffs. We first note that deciding the existence of PSNE in integer programming games with general polynomial-time computable payoffs is difficult even when the number of players and dimension of the polytopes are fixed:

Proposition 2.2.2. *Consider an instance $\langle n, P_1, \ldots, P_n, u_1, \ldots, u_n \rangle$ of an integer programming game with general polynomial-time-computable payoff given by the following input, in binary encoding,*

(I_1) *for each $i \in I = \{1, \ldots, n\}$, the dimension d_i and a binary encoding of the inequality description $(M_i, \mathbf{b}_i)$ of a rational polytope $P_i = \{\mathbf{x} \in \mathbb{R}^{d_i} : M_i\mathbf{x} \leq \mathbf{b}_i\}$;*

[3]Note that u_i could also be defined a rational-valued over some largest denominator L which would then become part of the input.

(I_2) *for each $i \in I$ a polynomial-time computable utility function u_i expressed as a Boolean circuit.*

Then, the problem of deciding if such a game possesses a PSNE *is Σ_2^P-complete. The same result holds if $n = 2$ and $d_i = 1$ for all $i \in I$.*

Proof. We show that the language

$$L = \{I : \text{ integer programming game instance } I = \langle n, P_1, \ldots, P_n, u_1, \ldots, u_n \rangle \text{ has a PSNE}\}$$

is in Σ_2^P by showing it satisfies Definition 1.4.1. Given an instance I, proof that I possesses a PSNE consists of an action profile $\mathbf{s} \in S$ and evidence that every possible unilateral deviation by a player does not lead to gains in utility for that player. That is,

$$I \in L \iff \exists \mathbf{s} \in S,\ \forall i\ \forall \mathbf{s}_i' \in S_i : u_i(\mathbf{s}) \geq u_i(\mathbf{s}_{-i}, \mathbf{s}_i').$$

Since our utilities are polynomial-time computable given strategy profile $\mathbf{s}$ and each alternative $(\mathbf{s}_{-i}, \mathbf{s}_i')$, this condition can be checked in polynomial time. Noting that the encoding of each action profile is polynomially bounded in the input size, this establishes that L is in Σ_2^P.

It remains to show that L is Σ_2^P-hard. We proceed by a reduction from circuit games (see Definition 1.1.1). Consider an instance of a 2-player circuit game specified by integers k_1, k_2 and circuits C_1, C_2 such that the action set $A_i \subseteq \{0,1\}^{k_i}$ and utility $\pi_i(\mathbf{a}) = C_i(\mathbf{a})$ for $\mathbf{a} = (\mathbf{a}_1, \mathbf{a}_2) \in A_1 \times A_2$. We transform this into an integer programming game instance $\langle 2, P_1, P_2, u_1, u_n \rangle$ where $P_i \subseteq \mathbb{R}$ for $i = 1, 2$.

An action profile $\mathbf{a} = (\mathbf{a}_1, \mathbf{a}_2)$ in the circuit game instance with $\mathbf{a}_i = (a_{il} \ldots, a_{ik_i}) \in \{0,1\}^{k_i}$ can be represented as action $s_i \in S_i$ with $s_i = a_{il} \cdots a_{ik_i}$ in binary expansion. Thus the polytope defining the action set of player i can be defined as $P_i = \{s_i \in \mathbb{R} : 0 \leq s_i \leq 2^{k_i}\}$.

Note that the binary encoding size of the polytope is $\log_2 2^{k_i} = k_i$. Let utility function $\pi_i(\mathbf{a}) = u_i(\mathbf{s})$ when $\mathbf{a} \in A_1 \times A_2$ for $i = 1, 2$ and $\mathbf{s}$ its corresponding binary expansion. Let $u_i(\mathbf{s}) = -\infty$ if the binary encoding of $\mathbf{s}$ is not in $A_1 \times A_2$. It is clear that this transformation can be performed in polynomial time. It is also clear that the circuit game possesses a PSNE if and only if its transformation as an integer programming game does as well. □

Due to the difficulty of the case where utilities are general polynomial-time-computable functions, we explore the following special case:

Definition 2.2.3 (**DPLC payoffs**)**.** An integer-valued payoff function $u : S \to \mathbb{Z}$ of a game is a *difference of piecewise-linear convex functions* or DPLC function if it can be expressed as:

$$u(\mathbf{s}) = \max_{k \in K} f_k(\mathbf{s}) - \max_{l \in L} g_l(\mathbf{s})$$

where the f_k and g_l are affine functions with integer coefficients and where K and L are finite index sets. We refer to the f_k as the "convex" pieces of u and the g_l as the "concave" pieces of u.

We consider integer programming games where each player i has a DPLC payoff function

$$u_i(\mathbf{s}) = \max_{k \in K_i} f_{ik_i}(\mathbf{s}) - \max_{l \in L_i} g_{il_i}(\mathbf{s}). \tag{2.2}$$

A first comment regarding this class of games is that they can express any finite game given in normal form. A normal form game is defined by action sets $A_1, \ldots, A_n$ and payoffs $\pi_i(\mathbf{a})$ for $\mathbf{a} \in A_1 \times \cdots \times A_n$ and $i \in I$. We refer to the set $A = \prod_{i=1}^n A_i$ as the set of action profiles. A normal form game is finite if A is finite. We use the alternate notation A_i and π_i – as opposed to S_i and u_i – for normal form games to draw a contrast between the fact that in normal form game the actions sets and payoffs for each action profile are given explicitly as part of the input, whereas in integer programming games the action sets S_i and payoffs

u_i are given implicitly.

The following result shows that integer programming games with DPLC payoffs are fully expressive of normal form games:

Proposition 2.2.4. *Every finite normal form game is equivalent to an integer programming game with* DPLC *payoffs.*

Proof. Let finite normal form game G be defined by action sets $A_1, \ldots, A_n$ and payoffs $\pi_i(\mathbf{a})$ for $\mathbf{a} \in A$. Let d_i equal the number of elements in the action set A_i. Let vector $\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,d_i})$ denote a mixed strategy profile for player i. Mixed strategy profiles $\mathbf{x}$ lie in the unit simplex:

$$P_i = \{\mathbf{x}_i = (x_{i,1}, \ldots, x_{i,d_i}) \in \mathbb{R}^{d_i} : \mathbf{x}_i \geq 0, \ \sum_{j=1}^{d_i} x_{i,j} = 1\}.$$

This is a polytope, and the lattice points inside P_i are exactly its vertices, which correspond to pure strategies (actions) for player i. An action a_i in A_i is represented by its characteristic vector $\chi_i(a_i) = (0, \ldots, 0, 1, 0, \ldots, 0)$ where the 1 is the in the ith position. Thus, we represent the set of pure strategies for player i as $S_i = P_i \cap \mathbb{Z}^{d_i}$. As above, let $S = S_1 \times \cdots \times S_n$. This conforms to the framework of integer programming games.

As for the payoffs, our goal is to convert the explicitly given payoffs π_i over the set A of action profiles into equivalent DPLC payoff functions u_i over S. Let $A_i^+ = \{\mathbf{a} \in A : \pi_i(\mathbf{a}) > 0\}$ and $A_i^- = \{\mathbf{a} \in A : \pi_i(\mathbf{a}) < 0\}$. For every $\mathbf{x} = (\chi_1(a_1), \ldots, \chi_n(a_n)) \in S$, define the DPLC payoff

$$u_i(\mathbf{x}) = \max_{a_i \in A_i^+} \pi_i(\mathbf{a}) \left(\sum_{i=1}^{n} x_{i,a_i} - n + 1 \right) - \max_{a_i \in A_i^-} \pi_i(\mathbf{a}) \left(n - \sum_{i=1}^{n} x_{i,a_i} - 1 \right).$$

Note that for every $\mathbf{a} \in A$ we have $u_i(\chi_1(a_1), \ldots, \chi_n(a_n)) = \pi_i(\mathbf{a})$. Thus the normal form game G is equivalent to the integer programming game with pure strategy sets S_i and DPLC

payoffs u_i. □

Note that this representation has the same input size as the normal form game itself. Note that every normal form game can be expressed as an integer programming game with DPLC payoffs, which expresses the universality of the integer programming game framework. Indeed, every game can be expressed as a normal form, and in turn in integer programming game, the catch being that not every game has a *compact* representation as an integer programming game. Indeed, some functional forms of utility may require an exponential number of pieces to represent exactly. Further computational complexity consequences of this proposition are discussed in Remark 2.3.12.

Another useful result demonstrates the power of modeling payoffs as DPLC functions. It can be shown that every continuous piecewise linear function can be represented as the difference of two piecewise linear convex functions [86, Theorem 4.2]. Thus, a DPLC function can be used to describe any continuous piecewise linear payoff function, which in turn can be used to approximate an arbitrary continuous payoff function. The caveat here is that many pieces may be required even for approximations, and as our algorithms will be sensitive to the number of pieces in the DPLC representation, this may diminish efficiency gains.

Example 2.2.5 (**Cournot game with indivisible goods and fixed production costs**)**.**

A set of n manufacturers (the players) produce indivisible goods which are then sold in a market. Player i chooses an integer production level s_{ij} for each of its indivisible products $j \in \{1, \ldots, d_i\}$ subject to resource constraints $M_i \mathbf{s}_i \leq \mathbf{b}_i$ where $\mathbf{s}_i = (s_{i1}, \ldots, s_{id_i})$. Thus, player i's action set S_i is the set of integer points in a polytope in $\mathbb{R}^{d_i}$. The payoff to each player consists of revenues and production costs. Under usual assumptions, the revenue to each player i is a concave function of the action profile $\mathbf{s}$, which can be expressed as a piecewise linear concave function $\min_{k \in R_i} r_{ik}(\mathbf{s})$. For each player i and product j there

may be a fixed production cost F_{ij}. The variable production cost is a convex function of the production levels $\mathbf{s}_i$ expressed as a piecewise linear convex function $\max_{l \in C_i} c_{il}(\mathbf{s}_i)$. The payoff to player i is thus

$$u_i(\mathbf{s}) = \sum_{j=1}^{d_i} \max\{-F_{ij}, -F_{ij}s_{ij}\} - \left(\max_{k \in R_i}(-r_{ik}(\mathbf{s})) + \max_{l \in C_i} c_{il}(\mathbf{s}_i) \right)$$

which can be expressed as a DPLC function, by distributing the max operator over the sums. For instance we can rewrite $\max_{l \in R_i}(-r_{il}(\mathbf{s})) + \max_{l \in C_i} c_{il}(\mathbf{s}_i)$ as

$$\max_{\{(l,k) \in R_i \times C_i\}} \{(-r_{il}(\mathbf{s}) + c_{il}(\mathbf{s}_i)\}$$

which is a piecewise linear convex function. As will be discussed further in Remark 2.3.13, this is precisely the structure that is analyzed in this chapter.

2.3 Computing pure Nash equilibria in integer programming games

Consider an integer programming game with DPLC payoffs as defined in Section 2.2. Our goal is to encode the PSNE of such a game as a short rational generating function. The most general setting we provide an efficient algorithm for such an encoding is when the number of players and the dimension of their action sets are fixed and each player's DPLC payoff function has the form:

$$u_i(\mathbf{s}) = \max_{k \in K_i} f_{ik}(\mathbf{s}) - \max_{l \in L_i} g_{il}(\mathbf{s}) \tag{2.3}$$

where we now assume that the size of K_i is a fixed. Since S is bounded we assume without loss of generality that $u_i(\mathbf{s}) \geq 0$ for all $i \in I$ and $\mathbf{s} \in S$. The analysis proceeds with two fundamental insights. First, when there are a fixed number of "convex" pieces, i.e.,

when $|K_i|$ is fixed, each player's payoff is piecewise linear concave within the region where a particular f_{ik} remains maximal. The second insight is that when payoffs are piecewise linear concave the hypograph of the payoff function is a polyhedral set, encodable as a short rational generating function.

We begin with a simple result towards partitioning the action profile space into regions according to the values of the linear pieces of the payoffs. We assume that K_i is a totally ordered set.

Lemma 2.3.1. *For each player i, the set of all action profiles can be expressed as a disjoint union*

$$S = \biguplus_{k \in K_i} S_{ik}$$

where

$$S_{ik} = \left\{ \mathbf{s} \in S : \begin{array}{ll} f_{ik}(\mathbf{s}) \geq f_{ij}(\mathbf{s}), & j > k \\ f_{ik}(\mathbf{s}) > f_{ij}(\mathbf{s}), & j < k \end{array} \right\}.$$

Proof. We first show that $S = \bigcup_{k \in K_i} S_{ik}$ and later establish it is a disjoint union. Clearly $\bigcup_{k \in K_i} S_{ik} \subseteq S$. It remains to show the reverse inclusion. Let $\mathbf{s} \in S$ and define $J(\mathbf{s}) = \{j \in K_i : f_{ik}(\mathbf{s}) = \max_{k \in K_i} f_{ik}(\mathbf{s})\}$ and $j(\mathbf{s}) = \min J(\mathbf{s})$, then $\mathbf{s} \in S_{ij}(\mathbf{s})$.

To show that the union is in fact disjoint suppose by contradiction that there exists an $\mathbf{s}$ in $S_{ik} \cap S_{ik'}$ where $k > k'$. If $k > k'$ then since $\mathbf{s} \in S_{ik}$ this implies $f_{ik}(\mathbf{s}) \geq f_{ik'}(\mathbf{s})$. However, since $\mathbf{s} \in S_{ik'}$ this yields that $f_{ik'}(\mathbf{s}) > f_{ik}(\mathbf{s})$, a contradiction. The result follows. □

Note that we could equivalently write S_{ik} as follows,

$$S_{ik} = \left\{ \mathbf{s} \in S : \begin{array}{ll} f_{ik}(\mathbf{s}) \geq f_{ij}(\mathbf{s}), & j > k \\ f_{ik}(\mathbf{s}) \geq f_{ij}(\mathbf{s}) + 1, & j < k \end{array} \right\}. \tag{2.4}$$

We can do this because the action profiles in S are integer vectors and the data defining the functions f_{ij} are integral. The same holds true for all the strict inequalities in this chapter,

and thus the alternative descriptions using a strict inequality or a weak inequality with one unit added to the larger side of the inequality are used interchangeably throughout.

The next step is to refine the partition of S to account for all players simultaneously. To do so we introduce the following convenient notation. Let $\mathbf{K} = \prod_{i=1}^{n} K_i$ be the set of all vectors $\mathbf{k} = (k_1, \ldots, k_n)$ of indices where $k_i \in K_i$ for $i \in I$. Using this notation, denote by $S_{\mathbf{k}}$ the intersection $S_{1k_1} \cap \cdots \cap S_{nk_n}$. Employing this notation we state the following simple corollary of the previous lemma:

Corollary 2.3.2. *The set of action profiles can be partitioned as follows:*

$$S = \biguplus_{\mathbf{k} \in \mathbf{K}} S_{\mathbf{k}}$$

Thus each action profile $\mathbf{s}$ lies in a unique subset $S_{\mathbf{k}}$ of S where it is known that the payoff for each player i is

$$u_i(\mathbf{s}) = f_{ik_i}(\mathbf{s}) - \max_{l \in L_i} g_{il}(\mathbf{s})$$

and hence a piecewise linear concave function of $\mathbf{s}$. To take advantage of this concave structure in the payoff we propose an extension of the game.

Definition 2.3.3 (**Extended Game**)**.** Given an integer programming game with DPLC payoffs let $\hat{\mathbf{s}} = (\mathbf{s}; \mathbf{y}) = (\mathbf{s}_1, \ldots, \mathbf{s}_n; y_1, \ldots, y_n)$ denote an *extended action profile* which includes variables $y_i \in \mathbb{Z}$ which keep track of payoff values. The *set $\hat{S}$ of extended action profiles* is the set

$$\hat{S} = \biguplus_{\mathbf{k} \in \mathbf{K}} \hat{S}_{\mathbf{k}} \tag{2.5}$$

where

$$\hat{S}_{\mathbf{k}} = \{\hat{\mathbf{s}} = (\mathbf{s}, \mathbf{y}) : \mathbf{s} \in S_{\mathbf{k}} \text{ and } 0 \leq y_i \leq f_{ik}(\mathbf{s}) - g_{il}(\mathbf{s}) \text{ for all } l \in L_i \text{ and } i \in I\}. \tag{2.6}$$

By Corollary 2.3.2 it is easy to see that (2.5) is a disjoint union.

We define the *extended utility function* $\hat{u}_i$ for each player i as

$$\hat{u}_i(\hat{\mathbf{s}}_i) = y_i$$

which is a linear function on $\hat{S}$. We call the tuple $\hat{G} = (\hat{S}, \hat{u}_1, \ldots, \hat{u}_n)$ the *extended game* of the original game $G = (S, u_1, \ldots, u_n)$.

At this point there are three important observations regarding the definition of extended games. Note that since S is a bounded by a polytope and the constraints $0 \leq y_i \leq f_{ik}(\mathbf{s}) - g_{il}(\mathbf{s})$ for all $l \in L_i$ and $i \in I$ bound the values of y_i, it follows that each $\hat{S}_{\mathbf{k}}$, and thus $\hat{S}$, is bounded. This fact will be important when encoding $\hat{S}$ by a rational generating function and applying the Linear Optimization Lemma (Lemma 1.4.10), which we have occasion to do in what follows. Note also that $\hat{u}_i(\hat{\mathbf{s}}_i)$ is a linear function over $\hat{S}$ and thus more amenable to encoding by generating functions than the piecewise linear payoffs of the original game.

We also remark that the extended game $\hat{G}$ is not a simultaneous-move game since the players' choices are not independent and some choices of $\hat{\mathbf{s}}$ may lead to infeasibility. Similar issues are explored in [11]; however, we do not treat the extended game $\hat{G}$ as a game unto itself, but simply as a mathematical construct to study the equilibria of the original game G, and we thus ignore these considerations.

A key step is to establish a correspondence between the equilibria of the extended game to those of the original game. As will be seen, this correspondence relies on the fact that the descriptions of the action profile sets involve disjoint unions.

In our analysis we often consider deviations from the vector $\mathbf{k} = (k_1, \ldots, k_n)$ in which the i-th component k_i is replaced with $k_i' \in K_i$. For convenience, let $\mathbf{k}_{-i}$ denote the vector

of the remaining (unchanged) indexes; therefore,

$$(\mathbf{k}_{-i}, k_i') = (k_1, \ldots, k_{i-1}, k_i', k_{i+1}, \ldots, k_n).$$

We follow a similar convention in terms of action profiles. Namely, given an action profile $\mathbf{s} = (\mathbf{s}_1, \ldots, \mathbf{s}_n)$, if player i deviates to action $\mathbf{s}_i'$ then we let $\mathbf{s}_{-i}$ denote the (unchanged) actions of the remaining players, so

$$(\mathbf{s}_{-i}, \mathbf{s}_i') = (\mathbf{s}_1, \ldots, \mathbf{s}_{i-1}, \mathbf{s}_i', \mathbf{s}_{i+1}, \ldots, \mathbf{s}_n).$$

The equilibrium concept used in the extended game is defined as follows:

Definition 2.3.4 (**Extended pure Nash equilibrium**). Let $\hat{G} = (\hat{S}, \hat{u}_1, \ldots, \hat{u}_n)$ be an extended game. An *extended pure Nash equilibrium* (extended PSNE) $\hat{\mathbf{s}} = (\mathbf{s}; \mathbf{y})$ is an extended action profile where if $\hat{\mathbf{s}} \in \hat{S}_{\mathbf{k}}$ then there does not exist $\hat{\mathbf{s}}_i' = (\mathbf{s}_i', y_i')$ such that $(\hat{\mathbf{s}}_{-i}, \hat{\mathbf{s}_i}') \in \hat{S}_{(\mathbf{k}_{-i}, k_i')}$ and $\hat{u}_i(\hat{\mathbf{s}}_{-i}, \hat{\mathbf{s}_i}') > \hat{u}_i(\mathbf{s})$.

Lemma 2.3.5. *Consider the game $G = (S, u_1, \ldots, u_n)$ and its extended game $\hat{G} = (\hat{S}, \hat{u}_1, \ldots, \hat{u}_n)$ as defined above.*

(i) *An extended* PSNE *of $\hat{G}$ must be of the form*

$$\hat{\mathbf{s}} = (\mathbf{s}; \mathbf{u}(s)) = (\mathbf{s}_1, \ldots, \mathbf{s}_n; u_1(\mathbf{s}), \ldots, u_n(\mathbf{s})). \tag{2.7}$$

(ii) *There is a bijection between the set N of* PSNE *of the original game and the set $\hat{N}$ of extended* PSNE *of the extended game.*

Proof. (i) Let $\hat{\mathbf{s}} = (\mathbf{s}; \mathbf{y}) \in \hat{S}$. By the disjoint union (2.5) there exists a unique $\mathbf{k}$ such that $\hat{\mathbf{s}} \in \hat{S}_{\mathbf{k}}$. It follows that $\mathbf{s} \in S_{\mathbf{k}}$. For all $i \in N$ we have $\mathbf{s} \in S_{ik_i}$ and $y_i \leq f_{ik_i}(\mathbf{s}) - \max_{l \in L_i} g_{il}(\mathbf{s}) = u_i(s)$. Thus, $\hat{u}_i(\hat{\mathbf{s}}) = y_i \leq u_i(\mathbf{s}) = \hat{u}_i(\mathbf{s}_1, u_1(\mathbf{s}); \ldots; \mathbf{s}_n, u_n(\mathbf{s}))$. Hence, whenever $y_i < u_i(\mathbf{s})$ it is

profitable for player i to deviate to the extended action $(\mathbf{s}_i; u_i(\mathbf{s}))$. Therefore, an extended PSNE must have the form (2.7). (ii) Consider the mapping $\varphi : N \longrightarrow \hat{N}$ defined by $\mathbf{s} \longmapsto (\mathbf{s}; u(\mathbf{s}))$. We claim φ is a well-defined bijection. First we show that φ is well-defined, that is $\varphi(\mathbf{s}) \in \hat{N}$ for all $\mathbf{s} \in N$. Clearly $\varphi(\mathbf{s})$ is in $\hat{S}$. Let $\mathbf{s} \in S_{\mathbf{k}}$ for some $\mathbf{k}$. Now consider a feasible deviating action for player i, $\hat{\mathbf{s}}'_i = (\mathbf{s}'_i, y'_i)$, where $(\hat{\mathbf{s}}_{-i}, \hat{\mathbf{s}}'_i) \in \hat{S}_{(\mathbf{k}_{-i}, k'_i)}$ for some $k'_i \in K_i$. In other words, player i deviates from choosing $\mathbf{s}_i \in S_{ik_i}$ to choosing $\mathbf{s}'_i \in S_{ik'_i}$ and changing y_i to y'_i. We have

$$\hat{u}_i(\hat{\mathbf{s}}_{-i}, \hat{\mathbf{s}}'_i) = y'_i \leq u_i(\mathbf{s}_{-i}, \mathbf{s}'_i) \leq u_i(\mathbf{s}) = \hat{u}_i(\hat{\mathbf{s}})$$

where the second inequality holds since $\mathbf{s}$ is a PSNE for the game G and the final equality follows from (i). It follows that $\hat{\mathbf{s}}$ is an extended PSNE for the game $\hat{G}$ and mapping φ is well-defined.

It is clear that φ is injective. As for surjectivity, by part (i) it follows that every PSNE in $\hat{N}$ has the form $(\mathbf{s}; u(\mathbf{s}))$ for some $\mathbf{s} \in S$. It just remains to show that all such equilibria arise with $\mathbf{s} \in N$. Suppose the contrary, that is $\hat{\mathbf{s}} = (\mathbf{s}; u(\mathbf{s})) \in \hat{N}$ but $\mathbf{s} \notin N$. Then there must be a profitable deviation $\mathbf{s}'_i \in S_i$ for some player i in the original game G; $u_i(\mathbf{s}_{-i}, \mathbf{s}'_i) > u_i(\mathbf{s})$. This implies that there is a profitable deviation $\hat{\mathbf{s}}'_i = (\mathbf{s}'_i; u_i(\mathbf{s}_{-i}, \mathbf{s}'_i))$ in the extended game since: $\hat{u}_i(\hat{\mathbf{s}}) = u_i(\mathbf{s}) < u_i(\mathbf{s}_{-i}, \mathbf{s}'_i) = \hat{u}_i(\hat{\mathbf{s}}_{-i}, \hat{\mathbf{s}}'_i)$. □

Example 2.3.6. To clarify the above concepts we consider a very simple example of an integer programming game and its extension. In fact, all the concepts can be exemplified with an example of a game with a single player. Consider the following payoff function which a single player attempts to optimize:

$$u(s) = f(s) - g(s)$$

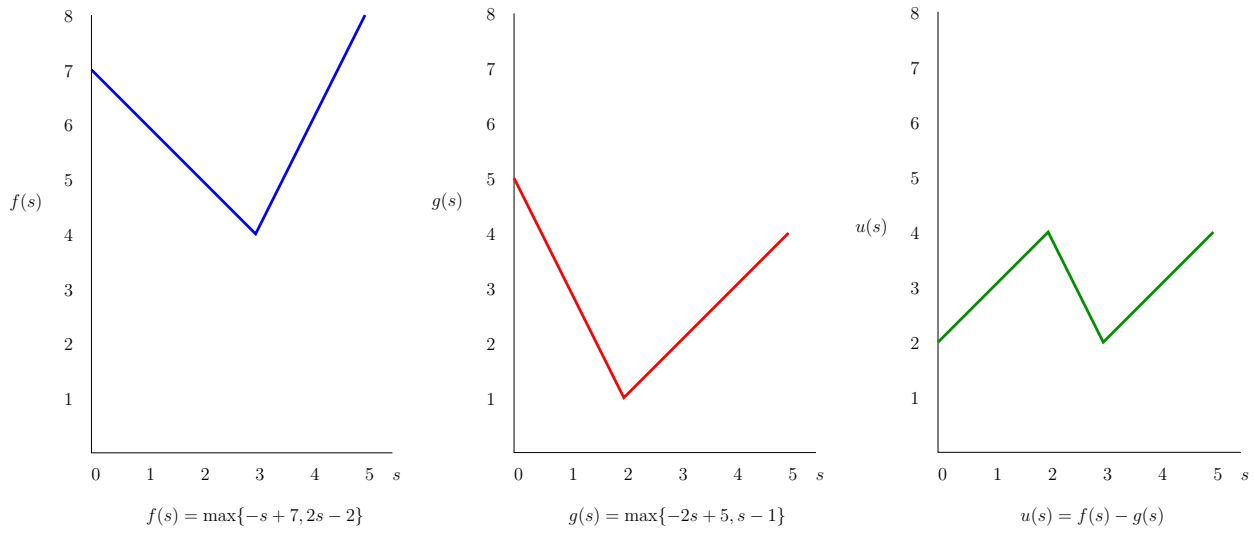


Figure 2.1: Payoff function for Example 2.3.6

over the set $S = \{0, 1, 2, 3, 4, 5\}$ where

$$f(s) = \max\{f_1(s), f_2(s)\} = \max\{-s + 7, 2s - 2\}$$

and

$$g(s) = \max\{g_1(s), g_2(s)\} = \max\{-2s + 5, s - 1\}.$$

From Figure 2.3.6 we see that

$$u(s) = \begin{cases} 2s + 2, & 0 \leq s \leq 2 \\ -2s + 8, & 2 \leq s \leq 3 \\ s - 1, & 3 \leq s \leq 5 \end{cases}$$

It is then straightforward to describe our sets of interest:

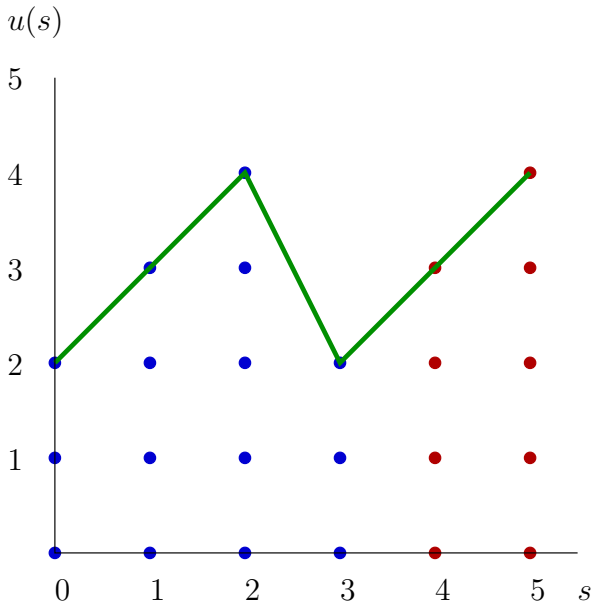


Figure 2.2: Graphical representation of the extended game for Example 2.3.6

$$
\begin{aligned}
S_1 &= \{s \in S : f_1(s) \geq f_2(s)\} = 0, 1, 2, 3 \\
S_2 &= \{s \in S : f_2(s) > f_1(s)\} = \{4, 5\} \\
\hat{S}_1 &= \{\hat{s} = (s, y) : s \in S_1, 0 \leq y \leq f_1(s) - g_1(s), 0 \leq y \leq f_1(s) - g_2(s)\} \\
&= (\{0, 1, 2, 3\} \times 0, 1, 2) \cup \{(1, 3), (2, 3), (2, 4)\} \\
\hat{S}_2 &= \{\hat{s} = (s, y) : s \in S_1, 0 \leq y \leq f_2(s) - g_1(s), 0 \leq y \leq f_2(s) - g_2(s)\} \\
&= (\{4, 5\} \times 0, 1, 2) \cup \{(4, 3), (5, 3), (5, 4)\} \\
\hat{S} &= \hat{S}_1 \cup \hat{S}_2
\end{aligned}
$$

Figure 2.3.6 gives a graphical representation. The lattice points above the interval $[0, 3]$ are the elements of the set $\hat{S}_1$ (in blue). Likewise, the lattice points above the interval $[4, 5]$ are the set $\hat{S}_2$ (in red). In this simple setting it is straightforward to illustrate Lemma 2.3.5. An equilibrium in the original game is an optimal solution to $\max\{u(s) : s \in S\}$. By inspection we see that the set of equilibria of the game is $N = \{2, 5\}$. An equilibrium in the extended game solves $\max\{y : (s, y) \in \hat{S}\}$. Again by inspection we see that $\hat{N} = \{(2, 4), (5, 4)\}$ is

the set of equilibria in the extended game. It is straightforward to see that N and $\hat{N}$ are in bijection under the mapping $\varphi(s, y) = s$.

We can now state the main result of the chapter:

Theorem 2.3.7. *Consider an integer programming game with* DPLC *payoffs given by the following input in binary encoding:*

(I_1) *a bound* $B \in \mathbb{N}$*;*

(I_2) *for each* $i \in I = \{1, \ldots, n\}$*, the dimension* d_i *and an inequality description* $(M_i, \mathbf{b}_i)$ *of a rational polytope* $P_i = \{\mathbf{x} \in \mathbb{R}^{d_i} : M_i\mathbf{x} \leq \mathbf{b}_i\} \subseteq [-B, B]^{d_i}$*;*

(I_3) *for each* $i \in I$*, nonnegative integers* $|K_i|$ *and* $|L_i|$*, and for all integers* k*,* l *such that* $1 \leq k \leq |K_i|$ *and* $1 \leq l \leq |L_i|$*, integer vectors* $\boldsymbol{\alpha}_{ik} \in \mathbb{Z}^d$*,* $\boldsymbol{\gamma}_{il} \in \mathbb{Z}^d$ *(where* $d = d_1 + \cdots + d_n$*) and integers* β_{ik}*,* δ_{ik} *defining the affine functions* $f_{ik} : S \to \mathbb{Z}$ *and* $g_{il} : S \to \mathbb{Z}$ *by* $f_{ik}(\mathbf{s}) = \boldsymbol{\alpha}_{ik} \cdot \mathbf{s} + \beta_{ik}$ *and* $g_{il}(\mathbf{s}) = \boldsymbol{\gamma}_{il} \cdot \mathbf{s} + \delta_{il}$ *for all* $\mathbf{s} \in S = \prod_{i=1}^{n} (P_i \cap \mathbb{Z}^{d_i})$*.*

The set $\hat{N}$ *of extended* PSNE *of the extended game* $\hat{G} = (\hat{S}, \hat{u}_1, \ldots, \hat{u}_n)$ *has a short rational generating function encoding, which can be computed in polynomial time when the total dimension* d *and the sizes* $|K_i|$ *are fixed for all* $i \in I$*.*

Proof. We express the set of extended PSNE as follows:

$$\hat{N} = \hat{S} \setminus \bigcup_{i=1}^{n} D_i$$

where

$$D_i = \biguplus_{\mathbf{k} \in \mathbf{K}} \bigcup_{k_i' \in K_i} \mathrm{proj}_{\hat{\mathbf{s}}} \left\{ (\hat{\mathbf{s}}, \hat{\mathbf{s}}_i') : \hat{\mathbf{s}} \in \hat{S}_{\mathbf{k}},\ (\hat{\mathbf{s}}_{-i}, \hat{\mathbf{s}}_i') \in \hat{S}_{(\mathbf{k}_{-i}, k_i')},\ \hat{u}_i(\hat{\mathbf{s}}_{-i}, \hat{\mathbf{s}}_i') \geq \hat{u}_i(\hat{\mathbf{s}}) + 1 \right\} \quad (2.8)$$

Note that some of the projected sets may be empty.

The set D_i consists of those action profiles where player i has a profitable deviation. The description of D_i in (2.8) is a union over profitable deviations from one set in the partition of $\hat{S}$ to another. This description of $\hat{N}$ is easy to verify using the definition of extended PSNE.

We now establish that $\hat{N}$ can be encoded as a short rational generating function. First we claim $\hat{S}$ admits such an encoding. Consider the description of $\hat{S}$ given in (2.5). The sets S_{ik} are sets of lattice points inside rational polytopes (as can be readily seen from (2.4)) and thus encodable as short rational generating functions. This implies that $S_{\mathbf{k}} = S_{1k_1} \cap \cdots \cap S_{nk_n}$ is a polytopal lattice point set, and thus $\hat{S}_{\mathbf{k}}$ is a set of lattice points inside a polytope (see (2.6) noting that all the f_{ik} and g_{il} are affine). By Lemma 1.4.3 this implies that $\hat{S}_{\mathbf{k}}$ admits a short rational generating function encoding. By Lemma 1.4.5, it follows that $\hat{S}$ can be encoded as a short rational generating functions in polynomial time.

Note in addition that the sets to be projected in (2.8) are again sets of lattice points inside of rational polytopes, by observing that the extended payoffs functions are linear. By the Projection Lemma it follows that each set in the union can be encoded as a short rational generating function. Using again the Boolean Operations Lemma (Lemma 1.4.4) we conclude that each D_i, and thus $\hat{N}$, admit short rational generating function encodings which can be computed in polynomial time when the sizes of the sets K_i are fixed for all $i \in I$. We remark that the outer union in (2.8) (indexed by $\mathbf{k} \in \mathbf{K}$) is a disjoint union; thus by Lemma 1.4.5 its rational generating function can be computed by adding the rational generating functions of the parts, rather than using the construction in the Boolean Operations Lemma (Lemma 1.4.4). □

Corollary 2.3.8. *Consider an integer programming game with* DPLC *payoffs given by the same input as in Theorem 2.3.7. The set N of* PSNE *has a short rational generating function encoding which can be computed in polynomial time when the total dimension d and the sizes $|K_i|$ are fixed for all $i \in I$.*

Proof. By the previous theorem, we can encode the set of *extended* PSNE, $\hat{N}$, in polynomial

time. Using the bijective map φ given in the proof of Lemma 2.3.5 and noting Remark 1.4.6 we can use an appropriate monomial substitution in the rational generating function description of $\hat{N}$ to yield a short rational generating function encoding for N. □

The true power of the previous lemma lies in the fact that having a short rational encoding of a set allows for efficient counting, optimizing and enumerating procedures as discussed in Section 1.4.2. Using these techniques we can answer numerous questions of interest on the existence, uniqueness and structure of equilibria.

Corollary 2.3.9. *Consider an integer programming game with* DPLC *payoffs given by the same input as in Theorem 2.3.7. There is a polynomial time algorithm to compute the number of* PSNE*, when the total dimension d and the sizes $|K_i|$ are fixed for all $i \in I$. In addition, under the same assumptions there is a polynomial time algorithm to find a sample* PSNE*, when at least one exists.*

Proof. Given the short rational generating function encoding of N we calculate $|N|$ in polynomial time by the Counting Lemma (Lemma 1.4.8. If an PSNE exists, we can output one by running the polynomial-delay enumeration algorithm on N described in the Enumeration Lemma (Lemma 1.4.9) and terminating just after one PSNE has been generated. This can also be done in polynomial time. □

This corollary can be used to answer the question of whether there is a unique PSNE – simply check whether $|N| = 1$. The following result is also immediate by the Enumeration Lemma (Lemma 1.4.9).

Corollary 2.3.10. *Consider an integer programming game with* DPLC *payoffs given by the same input as in Theorem 2.3.7. There is a polynomial-space polynomial-delay algorithm to enumerate all the* PSNE *of the game when the total dimension d and the sizes $|K_i|$ are fixed for all $i \in I$.*

Remark 2.3.11. Considering the number of elements of the game we need to fix in Corollary 2.3.9 – fixed number of players, fixed dimension of the polytopes, fixed sizes of the K_i – one might ask if there is an alternate method to generating functions that might yield similar results. The key observation is that the action sets are described implicitly as lattice points in polytopes given by linear inequalities, and thus the number of actions for each player may be exponential in the input size. Thus, brute-force enumeration of all the action profiles in S is a computationally intractable approach to the problem. Note that although brute-force enumeration is intractable, we obtain information that may appear to require an enumerative approach; for instance, we can count all equilibria.

Remark 2.3.12. It was shown in Proposition 2.2.4 that every normal form game can be expressed as an integer programming game with DPLC payoffs. Note, however, that the dimensions of the action spaces are equal to number of corresponding actions in the normal form game. Indeed, using the notation of the proof of Proposition 2.2.4, we have $S_i \in \mathbb{Z}^{A_i}$. From a complexity point of view this representation is unsatisfactory. In Corollary 2.3.9 we require the dimension of the action spaces to be fixed, and thus we can only handle a fixed number of actions in the underlying normal form game. Normal form games with a fixed number of players and fixed number of actions are computationally uninteresting.

Remark 2.3.13. The Cournot game in Example 2.2.5 fits the assumptions of Corollary 2.3.9. We assume that the number n of players (manufacturers) is "small", i.e., fixed, in order for each manufacturer to have appreciable market power. We also assume that the total number d of products is small. The sets K_i have cardinality $O(2^{d_i})$, which is fixed, and thus the decomposition of S in Corollary 2.3.2 is comprised of a constant number of subsets. Since the algorithm in Corollary 2.3.9 scales polynomially with the sizes of the sets L_i, we can afford a variable number of "concave" pieces in the description of the payoff functions. These "concave" pieces are used to represent general concave revenue functions and the convex parts of the cost functions, when restricted to integer points.

2.4 Related computations

In addition to counting and enumerating PSNE, generating function techniques can be used to derive efficient algorithms for related computations for integer programming games. Several of these are discussed in the following subsections.

First, note that the encoding of the set of PSNE as a short rational generating function, being a compact representation, is useful for learning about the specific structure of a game's equilibria. For instance, a simple calculation suffices for deciding the existence of a PSNE where player i plays a given action $\bar{\mathbf{s}}_i$. Indeed, simply find the short rational generating function encoding of

$$N^{(\bar{\mathbf{s}}_i)} = N \cap \{\mathbf{s} \in S : \mathbf{s}_i = \bar{\mathbf{s}}_i\}.$$

in polynomial time under the same assumptions in the previous section.

Now some more sophisticated calculations.

2.4.1 Pareto optimality

Consider the question of finding Pareto optimal PSNE, if any exist, in an integer programming game with DPLC payoffs. To tackle this, we start by encoding the set of Pareto optimal action profiles of the game.

Theorem 2.4.1. *Consider an integer programming game with* DPLC *payoffs given by the same input as in Theorem 2.3.7. The set PO of Pareto optimal action profiles has a short rational generating function encoding, which can be computed in polynomial time when the total dimension d and the sizes $|K_i|$ are fixed for all $i \in I$.*

Proof. The proof proceeds as follows:

(i) Define $\widehat{PO}$ as the set of Pareto optimal points in the extended game and find a generating function encoding for $\widehat{PO}$.

(ii) Derive a bijection between PO and $\widehat{PO}$.

(iii) Use the generating function of $\widehat{PO}$ and the bijection to obtain the generating function of PO.

For part (i) consider the following decomposition of $\widehat{PO}$:

$$\begin{aligned}\widehat{PO} &= \{\hat{\mathbf{s}} \in \hat{S} : \nexists\, \hat{\mathbf{s}}' \in \hat{S} \text{ s.t.}(\hat{u}_j(\hat{\mathbf{s}}') \geq \hat{u}_j(\hat{\mathbf{s}}) \ \forall j \in I) \text{ and } (\hat{u}_i(\hat{\mathbf{s}}') > \hat{u}_i(\hat{\mathbf{s}}) \ \exists i \in I)\} \\ &= \hat{S} \setminus \bigcup_{i=1}^{n} PD_i\end{aligned}$$

where

$$PD_i = \bigcup_{\mathbf{k},\mathbf{k}' \in \mathbf{K}} \text{proj}_{\hat{\mathbf{s}}} \left\{ (\hat{\mathbf{s}}, \hat{\mathbf{s}}') : \hat{\mathbf{s}} \in \hat{S}_{\mathbf{k}}, \ \ \hat{\mathbf{s}}' \in \hat{S}_{\mathbf{k}'}, \ \ \hat{u}_j(\hat{\mathbf{s}}') \geq \hat{u}_j(\hat{\mathbf{s}}) \quad \text{for all } j \neq i, \ \ \hat{u}_i(\hat{\mathbf{s}}') \geq \hat{u}_i(\hat{\mathbf{s}}) + 1 \right\}$$

is the set of Pareto dominated points due to a better alternative for player i. By analogous arguments as in the proof of Theorem 2.3.7 we can encode $\widehat{PO}$ as a short rational generating function in polynomial time. This establishes (i).

As for (ii) the argument is nearly identical to that in the proof of Lemma 2.3.5 and is therefore omitted. Finally, (iii) uses the same idea as found in the proof Corollary 2.3.8. □

Using the Boolean Operations Lemma (Lemma 1.4.4) we obtain a short rational generating function encoding of the set $N \cap PO$ of all Pareto optimal PSNE of the original game:

Corollary 2.4.2. *Consider an integer programming game with* DPLC *payoffs given by the same input as in Theorem 2.3.7. The set of Pareto optimal* PSNE *has a short rational generating function encoding, which can be computed in polynomial time when the total dimension d and the sizes $|K_i|$ are fixed for all $i \in I$.*

As in Section 2.3 we can use this generating function encoding to count and enumerate Pareto optimal equilibria.

2.4.2 Pure prices of anarchy and stability

The pure price of anarchy of a game measures the negative effects of competition on social welfare. The *social welfare* of an action profile is the corresponding total payoff of all players. Let w^* denote the maximum social welfare attainable under cooperation of the players; that is,

$$w^* = \max \left\{ \sum_{i=1}^{n} u_i(\mathbf{s}) : \mathbf{s} \in S \right\}. \tag{2.9}$$

and $\tilde{w}$ denote the worst social welfare attained by a PSNE; that is,

$$\tilde{w} = \min \left\{ \sum_{i=1}^{n} u_i(\mathbf{s}) : \mathbf{s} \in N \right\}.$$

The pure price of anarchy is the ratio $\tilde{w}/w^*$. A related concept is the pure price of stability which is the ratio of the best social welfare attained by a PSNE to the maximum social welfare attainable under cooperation. The pure price of anarchy has been studied in various network games, see [35] and the references therein for recent results on weighted congestion games. Using rational generating function techniques we can calculate the pure price of anarchy and stability efficiently in our setting.

Theorem 2.4.3. *Consider an integer programming game with* DPLC *payoffs given by the same input as in Theorem 2.3.7. There exist algorithms to compute the pure price of anarchy and the pure price of stability that run in polynomial time when the total dimension d and the sizes $|K_i|$ are fixed for all $i \in I$.*

Proof. Again we work with the extended game. The first step in calculating w^* is to note

that

$$w^* = \max\left\{\sum_{i=1}^{n} y_i : (\mathbf{s};\mathbf{y}) \in \hat{S}\right\}. \tag{2.10}$$

The equivalence of (2.9) and (2.10) is verified by first noting that for all $(\mathbf{s};\mathbf{y}) \in \hat{S}$, $y_i \leq u_i(\mathbf{s}) = \sum_i \hat{u}_i(\mathbf{s}; u(\mathbf{s}))$. Therefore, if $(\mathbf{s}^*, \mathbf{y}^*)$ is an optimal solution to the right-hand side of (2.10) then we must have $y_i^* = u_i(\mathbf{s})$ for all $i \in N$. This implies $\mathbf{s}$ is an optimal solution the right-hand side of (2.9).

To find w^* we optimize the linear function $\sum_i y_i$ over $\hat{S}$. Every $(\mathbf{s}, \mathbf{y}) \in \hat{S}$ satisfies $\mathbf{s} \in [-B, B]^d$ and

$$y_i \leq u_i(\mathbf{s}) \leq B\left(\max_{k\in K_i}(||\boldsymbol{\alpha}_{ik}||_1 + |\beta_{ik}|) + \max_{l\in L_i}(||\boldsymbol{\gamma}_{il}||_1 + |\delta_{il}|)\right).$$

That is, $(\mathbf{s};\mathbf{y}) \in [-M, M]^{d+n}$ for polynomially sized integer M (the right hand side of the above expression). Since in Section 2.3 we found a short rational generating function encoding of $\hat{S}$, we apply the Linear Optimization Lemma (Lemma 1.4.10) to calculate w^* in polynomial time.

To calculate $\tilde{w}$ we note

$$\tilde{w} = \min\left\{\sum_{i=1}^{n} y_i : (\mathbf{s};\mathbf{y}) \in \hat{N}\right\},$$

which follows from Lemma 2.3.5(i). Using the short generating function encoding of $\hat{N} \subseteq \hat{S}$ found in Section 2.3 we again apply the Linear Optimization Lemma (Lemma 1.4.10) to calculate $\tilde{w}$ in polynomial time. Thus, we obtain the price of anarchy $w^*/\tilde{w}$ in polynomial time.

The method for calculating the pure price of stability is similar. □

2.4.3 Pure threat point

The *pure minmax value to player i* in a game is defined as:

$$\min_{\mathbf{s}_{-i} \in S_{-i}} \max_{\mathbf{s}_i \in S_i} u_i(\mathbf{s}_i, \mathbf{s}_{-i}). \tag{2.11}$$

Although mixed strategies are usually considered in calculating (mixed) minmax values, an important quantity in classical game theory, here we restrict attention to pure strategies. The vector of *mixed* minmax values is known as the (mixed) *threat point*, which has drawn recent attention in the study of repeated games and explorations of computational implications of the Folk Theorem (see [12]). Analogously, we consider the *pure threat point* as the vector of pure minmax values. The concept of pure threat point was previously discussed in the context of bounded rationality in [69].

It was recently shown that, in various restrictive settings, the problem of calculating the (mixed) threat point is NP-hard. For instance, it can be shown that computing the (mixed) threat point of a three player game with binary payoffs ($\{0,1\}$) is NP-hard to approximate (see [12] Theorem 1 for a precise statement and proof.) Despite this negative result, we show that pure threat points can be computed efficiently in our setting.

Theorem 2.4.4. *Consider an integer programming game with* DPLC *payoffs given by the same input as in Theorem 2.3.7. There exists a polynomial time algorithm to compute the* pure *threat point when the total dimension d and the sizes $|K_i|$ are fixed for all $i \in I$.*

Proof. We begin by demonstrating how to calculate the pure minmax value for player i in polynomial time for each player i. Observe that an optimal value to the following bilevel optimization problem is the pure minmax value of player i:

$$\min_{\mathbf{s}_i, \mathbf{s}_{-i}} \left\{ u_i(\mathbf{s}_i, \mathbf{s}_{-i}) : \mathbf{s}_{-i} \in S_{-i},\ \mathbf{s}_i \in \arg\max_{\mathbf{s}'_i \in S_i} u_i(\mathbf{s}'_i, \mathbf{s}_{-i}) \right\}. \tag{2.12}$$

This bilevel optimization problem (see [18]) has essentially two players: a lower level player, or follower, who is player i, and an upper level player, or leader, who represents all the other players cooperating to "punish" i. Let

$$G_i = \left\{\mathbf{s} \in S : \mathbf{s}_i \in \arg\max_{\mathbf{s}_i \in S_i} \{u_i(\mathbf{s}_i, \mathbf{s}_{-i})\}\right\}$$

denote the set of bilevel feasible solutions to (2.12). Note that (2.12) is equivalent to $\min\{u_i(\mathbf{s}) : \mathbf{s} \in G_i\}$.

As before we turn our attention to the extended game. We define the analogous set $\hat{H}_i$:

$$\hat{H}_i = \left\{\hat{\mathbf{s}} \in \hat{S} : \hat{\mathbf{s}}_i \in \arg\max_{\hat{\mathbf{s}}'_i} \{\hat{u}_i(\hat{\mathbf{s}}_{-i}, \hat{\mathbf{s}}_i) : (\hat{\mathbf{s}}_{-i}, \hat{\mathbf{s}}'_i) \in \hat{S}\}\right\}.$$

Observe that if $\hat{\mathbf{s}} = (\mathbf{s}, \mathbf{y}) \in \hat{H}_i$ then $y_i = u_i(\mathbf{s})$. The set $\hat{H}_i$ can be expressed as $\hat{H}_i = \hat{S} \setminus D_i$, where D_i is defined as in (2.8). This follows since the optimization problem facing player i is the same problem as when determining extended PSNE is a single player game.

Thus by a direct application of Theorem 2.3.7 we can encode $\hat{H}_i$ as a short rational generating function. Note that $\hat{H}_i \subseteq \hat{S} \subseteq [-M, M]^{d+n}$ where M is as defined in the proof of Theorem 2.4.3. By applying the Linear Optimization Lemma (Lemma 1.4.10) find the optimal value of $\min\{y_i : (\mathbf{s}, \mathbf{y}) \in \hat{H}_i\} = \min\{\{u_i(\mathbf{s}) : \mathbf{s} \in H_i\}$, in polynomial time under the stated assumptions. The pure threat point can thus be calculated in polynomial time by finding the minmax value for each player. □

2.5 Stackelberg–Nash equilibria

We now turn to applying these techniques to a sequential setting. In a *Stackelberg–Nash game*, Player 0 (the leader) chooses an action, described by a vector $\mathbf{s}_0 \in S_0$. The remaining players $i \in I = \{1, \ldots, n\}$ (the followers) then simultaneously choose their actions $\mathbf{s}_i \in S_i(\mathbf{s}_0)$.

Each player $i \in I_0 = \{0, 1, \ldots, n\}$ then collects a payoff $u_i(\mathbf{s}_0, \mathbf{s})$ where $\mathbf{s} = (\mathbf{s}_1, \ldots, \mathbf{s}_n)$.

We assume $S_0 = P_0 \cap \mathbb{Z}^{d_0}$ where P_0 is a rational polytope. For each $\mathbf{s}_0 \in S_0$ the followers play an integer programming game with DPLC payoffs. The action of each follower $i \in I$ is described by a vector $\mathbf{s}_i \in \mathbb{Z}^{d_i}$ from the set $S_i(\mathbf{s}_0) = P_i(\mathbf{s}_0) \cap \mathbb{Z}^{d_i}$. We assume $P_i(\mathbf{s}_0)$ is the rational polytope

$$P_i(\mathbf{s}_0) = \{\mathbf{x} \in \mathbb{R}^{d_i} : M_i\mathbf{x} \leq \pi_i(\mathbf{s}_0)\} \quad \text{for} \quad i \in I$$

where $\pi_i(\mathbf{s}_0)$ is an integer valued affine function. Let $d = d_1 + \cdots + d_n$ and $d^+ = d_0 + d$.

Regarding payoffs, we assume each follower has a DPLC payoff $u_i(\mathbf{s})$ – independent of the leader's choice $\mathbf{s}_0$ – and given by

$$u_i(\mathbf{s}) = \max_{k \in K_i} f_{ik}(\mathbf{s}) - \max_{l \in L_i} g_{il}(\mathbf{s}). \tag{2.13}$$

The leader's payoffs are defined as the DPLC function

$$u_0(\mathbf{s}_0, \mathbf{s}) = \max_{k \in K_0} f_{0k}(\mathbf{s}_0, \mathbf{s}) - \max_{l \in L_0} g_{0l}(\mathbf{s}_0, \mathbf{s}). \tag{2.14}$$

We assume all K_i and L_i are finite index sets and all f_{ik} and g_{il} are integer valued affine functions.

Observe that given $\mathbf{s}_0 \in S_0$ we have a setup identical to that of Section 2.3, where the set of action profiles for the followers is $S(\mathbf{s}_0) = \prod_{i=1}^{n} S_i(\mathbf{s}_0) \subseteq \mathbb{Z}^d$.

We are interested in computing an optimal action for the leader while guaranteeing there exists a PSNE between the followers. Let $N(\mathbf{s}_0)$ denote the set of PSNE between the followers when the leader has chosen action $\mathbf{s}_0 \in S_0$. As in Section 2.3, a PSNE in $N(\mathbf{s}_0)$ is an action profile $\mathbf{s} \in S(\mathbf{s}_0)$ such that for every $i \in I$ there does not exist a deviation $s_i' \in S_i(\mathbf{s}_0)$ such that $u_i(\mathbf{s}_{-i}, \mathbf{s}_i') > u_i(\mathbf{s})$.

The leader faces the following optimization problem:

$$\max_{\mathbf{s}_0,\mathbf{s}}\{u_0(\mathbf{s}_0,\mathbf{s}) : s_0 \in S_0 \text{ and } s \in N(\mathbf{s}_0)\}. \tag{2.15}$$

Let N^+ denote the set of all *optimistic Stackelberg–Nash equilibria*, i.e., optimal solutions $(\mathbf{s}_0;\mathbf{s})$ to the optimization problem (2.15).

Note that this formulation implicitly assumes that the leader, after choosing $\mathbf{s}_0$, can choose a PSNE $\mathbf{s} \in N(\mathbf{s}_0)$ in order to maximize her payoff. This is a generalization to the case of competing followers of the "optimistic assumption" common in the multilevel optimization literature. The simplest illustration of the assumption is in the bilevel setting where the leader has the ability to choose among alternate optima to the follower's problem (see [18]). Here we assume more generally that the leader can choose among the alternate PSNE between the followers. We thus refer to N^+ as the set of *optimistic* Stackelberg–Nash equilibria.

The focus solely on pure strategies may need some motivation. Some choice of $\mathbf{s}_0 \in S_0$ may give rise to no PSNE in the followers' game, leaving only mixed Nash equilibria. We assume that the leader will avoid such an $\mathbf{s}_0$, even if it gave rise to higher *expected* payoffs. One may consider this as an extreme form of risk aversion, where any PSNE is preferred by the leader so as to avoid any uncertainty in payoffs.

Dealing with the possibility that $N(\mathbf{s}_0)$ may contain more than one PSNE (for a given s_0) is an important issue in many applied game theory contexts. Cachon and Netessine [14] discuss the issue in the context of applied game theory in operations management (OM). In many applications in OM the focus is on finding settings where there is a single PSNE, in which case there are no mixed equilibria. Isolating attention to where there is a unique PSNE has the added benefit that in such cases it is easier to predict the outcomes of a game. Indeed, when there are multiple equilibria, the leader is unsure which will prevail,

and whether in fact the players will all coordinate their actions on the same PSNE. We return to this issue later in the section.

We now show how to encode the set of optimistic Stackelberg–Nash equilibria as a short rational generating function:

Theorem 2.5.1. *Consider a Stackelberg–Nash game with* DPLC *payoffs defined by the following input, given in binary encoding:*

(I_1) *the number n of followers, and a bound $B \in \mathbb{N}$;*

(I_2) *the dimension d_0 and an inequality description $(M_0, \mathbf{b}_0)$ of a rational polytope $P_0 = \{\mathbf{x} \in \mathbb{R}^{d_0} : M_0\mathbf{x} \leq \mathbf{b}_0\} \subseteq [-B, B]^{d_0}$ defining the leader's feasible set $S_0 = P_0 \cap \mathbb{Z}^{d_0}$;*

(I_3) *for each $i \in I = \{1, \ldots, n\}$, the dimension d_i, number m_i of constraints, integer $m_i \times d_i$ matrix M_i, integer $d_0 \times m_i$ matrix Φ_i and integer vector $\psi_i \in \mathbb{Z}^{m_i}$ defining the affine function $\pi_i : S_0 \to \mathbb{Z}^{m_i}$ by $\pi_i(\mathbf{s}_0) = \Phi_i\mathbf{s}_0 + \psi_i$, and defining the follower i's parameterized polytope $P_i(\mathbf{s}_0) = \{\mathbf{x} \in \mathbb{R}^{d_i} : M_i\mathbf{x} \leq \pi_i(\mathbf{s}_0)\}$;*

(I_4) *for each $i \in I$, nonnegative integers $|K_i|$ and $|L_i|$, and for all integers k, l such that $1 \leq k \leq |K_i|$ and $1 \leq j \leq |L_i|$, integer vectors $\boldsymbol{\alpha}_{ik} \in \mathbb{Z}^d$, $\boldsymbol{\gamma}_{il} \in \mathbb{Z}^d$ (where $d = d_1 + \cdots + d_n$) and integers β_{ik}, δ_{ik} defining the affine functions $f_{ik} : \mathbb{Z}^d \to \mathbb{Z}$ and $g_{il} : \mathbb{Z}^d \to \mathbb{Z}$ by $f_{ik}(\mathbf{s}) = \boldsymbol{\alpha}_{ik} \cdot \mathbf{s} + \beta_{ik}$ and $g_{il}(\mathbf{s}) = \boldsymbol{\gamma}_{il} \cdot \mathbf{s} + \delta_{il}$ for all $\mathbf{s} \in \mathbb{Z}^d$;*

(I_5) *nonnegative integers $|K_0|$ and $|L_0|$, and for all integers k, l such that $1 \leq k \leq |K_0|$ and $1 \leq j \leq |L_0|$, integer vectors $\boldsymbol{\alpha}_{0k} \in \mathbb{Z}^{d^+}$, $\boldsymbol{\gamma}_{0l} \in \mathbb{Z}^{d^+}$ (where $d^+ = d_0 + d$) and integers β_{0k}, δ_{0k} defining the affine functions $f_{0k} : \mathbb{Z}^{d^+} \to \mathbb{Z}$ and $g_{0l} : \mathbb{Z}^{d^+} \to \mathbb{Z}$ by $f_{0k}(\mathbf{s}_0, \mathbf{s}) = \boldsymbol{\alpha}_{ik} \cdot (\mathbf{s}_0, \mathbf{s}) + \beta_{0k}$ and $g_{0l}(\mathbf{s}_0, \mathbf{s}) = \boldsymbol{\gamma}_{0l} \cdot (\mathbf{s}_0, \mathbf{s}) + \delta_{0l}$ for all $(\mathbf{s}_0, \mathbf{s}) \in \mathbb{Z}^{d^+}$.*

Then there exists a polynomial-time algorithm to compute the leader's optimum payoff and a short rational generating function encoding of the set N^+ of all optimistic Stackelberg–Nash equilibria when the total dimension d^+ and the sizes $|K_0|, |K_1|, \ldots, |K_n|$ are fixed.

Proof. We mimic the development leading up to Theorem 2.3.7 in Section 2.3 by defining an extended game with extended strategy profiles $(\mathbf{s}_0, \mathbf{s}, \mathbf{y})$ where $y_i \leq u_i(\mathbf{s})$ for all $i \in I$. As before denote $\hat{\mathbf{s}} = (\mathbf{s}, \mathbf{y})$. Let

$$S_{ik} = \{(\mathbf{s}_0, \mathbf{s}) : \mathbf{s}_0 \in S_0,\ \mathbf{s} \in S(s_0),\ f_{ik}(\mathbf{s}) \geq f_{ij}(\mathbf{s}) \text{ for } \ j > k,\ f_{ik}(\mathbf{s}) > f_{ij}(\mathbf{s}) \text{ for } \ j < k\}$$

and thus construct the disjoint union $\hat{S} = \biguplus_{\mathbf{k} \in \mathbf{K}} \hat{S}_{\mathbf{k}}$ where

$$\hat{S}_{\mathbf{k}} = \{(\mathbf{s}_0, \mathbf{s}, \mathbf{y}) : \mathbf{s}_0 \in S_0,\ \mathbf{s} \in S_{\mathbf{k}} \text{ and } 0 \leq y_i \leq f_{ik}(\mathbf{s}) - g_{il}(\mathbf{s}) \text{ for all } l \in L_i \text{ and } i \in I\}.$$

denoting $S_{\mathbf{k}} = S_{1k_1} \cap \cdots \cap S_{nk_n}$. Note that $\hat{S}_{\mathbf{k}}$ is a lattice point set in a polytope, and thus we can encode $\hat{S}$ by a short rational generating function.

Let $\hat{N}$ denote the set of extended action profiles $(s_0; \hat{s})$ such that $s_0 \in S_0$ is any feasible leader action and $\hat{s}$ is a PSNE in the followers extended game when the leader has chosen s_0. Now express $\hat{N}$ as

$$\hat{N} = \hat{S} \setminus \bigcup_{i=1}^{n} D_i$$

where

$$D_i = \biguplus_{\mathbf{k} \in \mathbf{K}} \bigcup_{k_i' \in K_i} \text{proj}_{\mathbf{s}_0, \hat{\mathbf{s}}} \left\{ (\mathbf{s}_0, \hat{\mathbf{s}}, \hat{\mathbf{s}}_i') : \begin{array}{r} (\mathbf{s}_0, \hat{\mathbf{s}}) \in \hat{S}_{\mathbf{k}}, \\ (\mathbf{s}_0, \hat{\mathbf{s}}_{-i}, \hat{\mathbf{s}}_i') \in \hat{S}_{(\mathbf{k}_{-i}, k_i')}, \\ \hat{u}_i(\hat{\mathbf{s}}_{-i}, \hat{\mathbf{s}}_i') \geq \hat{u}_i(\hat{\mathbf{s}}) + 1 \end{array} \right\}.$$

Since all sets to be projected are lattice point sets inside polytopes, we can apply reasoning as in the proof of Theorem 2.3.7 and encode $\hat{N}$ by a short rational generating functions under the stated assumptions. We establish a bijection between $\hat{N}$ and the set $N = \{(\mathbf{s}_0, \mathbf{s}) : \mathbf{s}_0 \in S_0,\ \mathbf{s} \in N(\mathbf{s}_0)\}$ of feasible solutions to (2.15). We claim the function $\varphi : N \longrightarrow \hat{N}$ defined by $(\mathbf{s}_0, \mathbf{s}) \longmapsto (\mathbf{s}_0, \mathbf{s}; u(\mathbf{s}))$ is a well-defined bijection. The details are similar to that in the

proof of Lemma 2.3.5 and are thus omitted. Via this bijection we derive a short rational generating function encoding of N by monomial substitution.

Now, let $N^{(v)}$ denote the set of feasible solutions to (2.15) that guarantee the leader a payoff of at least v:

$$\begin{aligned} N^{(v)} &= N \cap (\{(\mathbf{s}_0, \mathbf{s}) : u_0(\mathbf{s}_0, \mathbf{s}) \geq v\}) \\ &= N \cap \left(\bigcup_{k \in K_0} \{(\mathbf{s}_0, \mathbf{s}) : f_{0k}(\mathbf{s}_0, \mathbf{s}) - g_{0l}(\mathbf{s}_0, \mathbf{s}) \geq v \quad \forall l \in L_0\} \right). \end{aligned}$$

Note this is an intersection and union of polytopal lattice point sets. Since the size of K_0 is fixed we can apply the Boolean Operations Lemma (Lemma 1.4.4) to encode $N^{(v)}$ by a short rational generating function. Now, using binary search for v between $-B^+$ and B^+ where

$$B^+ = B \left(\max_{k \in K_0} (||\boldsymbol{\alpha}_{0k}||_1 + |\beta_{0k}|) + \max_{l \in L_0} (||\boldsymbol{\gamma}_{0l}||_1 + |\delta_{0l}|) \right),$$

and the counting algorithm to test for non-emptiness of $N^{(v)}$, we can find an optimal payoff v^+ to the leader. The set N^+ is therefore equal to $N^{(v^+)}$. □

As in the previous section, a short rational generating function encoding of the set N^+ of Stackelberg–Nash equilibria leads to results analogous to Corollary 2.3.9 and Corollary 2.3.10. Thus, we can derive efficient procedures to decide on the existence of and to enumerate Stackelberg–Nash equilibria.

From our discussion above, we know that N^+ may contain some undesirable equilibria, in particular those $(\mathbf{s}_0, \mathbf{s})$ where there are multiple equilibria in $N(\mathbf{s}_0)$. We now demonstrate a procedure (Algorithm 1 below) to find those optimistic equilibria $(\mathbf{s}_0, \mathbf{s}) \in N^+$ where there is a unique PSNE in the follower's game; that is, $|N(\mathbf{s}_0)| = 1$. The input for Algorithm 1 is the same as in Theorem 2.5.1. The output is a set M containing all the equilibria of interest.

If we are interested in finding only a single element of M we could terminate the algorithm

Algorithm 1 Compute $(\mathbf{s}_0, \mathbf{s}) \in N^+$ with $N(\mathbf{s}_0) = 1$

```
M ← ∅
while N⁺ ≠ ∅ do
    Find (s₀, s) ∈ N⁺
    if |N(s₀)| = 1 then
        M ← M ∪ {(s₀, s)}
    end if
    N⁺ ← N⁺ \ {(s₀, s)}
end whilereturn M
```

once one is found. The following result demonstrates how each iteration of the algorithm can be performed in polynomial time using our generating function approach. This represents a significant speed-up over any naïve method, for instance, brute-force enumeration, since finding an element $(\mathbf{s}_0, \mathbf{s}) \in N^+$ and computing $|N(\mathbf{s}_0)|$ both appear to require exponential work. For instance, in general the size of $N(\mathbf{s}_0)$ may be exponential for each choice of s_0 and thus brute-force enumeration yields a count only after exponential work. As the following result demonstrates, generating functions allow us to tackle both computations efficiently using essentially the same ideas as in previous results of this chapter.

Proposition 2.5.2. *Each iteration of the "while" loop in the above algorithm can be performed in polynomial time given the same input and conditions and fixed parameters as in Theorem 2.5.1.*

Proof. We extend the enumeration algorithm from the Enumeration Lemma (Lemma 1.4.9) applied to the Barvinok encoding of N^+ by including three additional steps between the output of each element $(\mathbf{s}_0, \mathbf{s})$ of N^+. The first additional step is to apply Theorem 2.3.8 to find a Barvinok encoding of $N(\mathbf{s}_0)$. The second step is to apply the Counting Lemma (Lemma 1.4.8) to $N(\mathbf{s}_0)$ to calculate its cardinality. The third step is to add $(\mathbf{s}_0, \mathbf{s})$ into M if it is the unique element of $N(\mathbf{s}_0)$. Clearly each of these three additional steps can be executed in polynomial time. Since the enumeration algorithm is polynomial delay, this implies that each iteration of the "while" loop runs in polynomial time. □

2.6 Conclusions and directions for further research

In this chapter we introduced a class of games and proposed algorithms for studying their PSNE using rational generating functions. Integer programming games are games of exponential type with exponentially many actions in the input size, and thus difficult to solve in general. It turns out that a piecewise-linear structure can be expressive of general utilities and yet has appealing computational properties where the complexity of our algorithms grows in the input size of these utilities and not the number of actions.

The simplicity by which we used generating functions to compute important information on the structure of these games, demonstrates the power of generating functions as an analytical and computational tool for game theory. Indeed, this idea was the motivation for the next chapter of the thesis where our analysis will again underscore the usefulness of the generating function approach.

There is considerable scope to explore applications of integer programming games to other situations. In addition, the use of generating function techniques is a novel approach to game theory which we feel may be applied to various types of games; e.g., threat point computations for repeated integer programming games; games with other payoff functions; and computations related to other solution concepts.

Another direction one might consider is to search for special structure on the polyhedral action sets; for instance, knapsack constraints which might yield structure on the set of equilibria. For example, Hemmeke, Onn and Weismantel consider a special class in integer programming games where it can be shown that generalized Nash equilibria can be computed in polynomial time even in varying dimension [44]. Some other directions for exploration can be found in several papers that have explored continuous analogues of integer programming games: polytope games [11], games on polyhedra [75] and games with polyhedral domain [65]. These authors use the theory of linear programming to characterize and identify properties

of equilibria and in some cases reduce to the problem of computing equilibria to the solving of linear programs. We feel that by using the theory of integer programming even more can be said about games whose data includes integer points inside of polyhedra.

Chapter 3

Symmetric games with piecewise linear utilities

3.1 Introduction

We continue our pursuit of understanding the structure of pure strategy Nash equilibria (PSNE) in a special class of games that admit compact representations: symmetric games. As mentioned in Chapter 1 the problem of computing PSNE of compactly-represented games is hard in the most general case, when utility functions are arbitrary efficiently-computable functions represented as Turing Machines [2] or circuits [73]. Researchers have studied compact game representations that exploit various types of structure in utility functions. One important type of structure is symmetry and is the focus of this chapter.

A game is *symmetric* when all players are identical and interchangeable. Symmetric games have been studied since the beginning of noncooperative game theory. For example, Nash proved that symmetric games always have a symmetric mixed Nash equilibrium [63]. In a symmetric game, a player's utility depends only on the player's chosen action and the *configuration*, which is the vector of integers specifying the number of players choosing each of the actions. As a result, symmetric games can be represented more compactly than games in normal form: we only need to specify a utility value for each action and each configuration. For a symmetric game with n players and m actions per player, the number of configurations is $\binom{n+m-1}{m-1}$. With fixed m, this grows like n^{m-1}, and $\Theta(n^m)$ numbers are

required to specify the game. Questions about PSNE can be answered straightforwardly by checking all configurations, which requires polynomial time in the size of the representation, and polynomial time in n when m is fixed [13].

Existing work on symmetry in games focuses on utility functions that explicitly enumerate utility values for each configuration. However, more concise representations are possible when utility functions have additional structure. In symmetric games, the set of players can be specified implicitly by the integer n, requiring only $\log n$ bits to represent. If utility functions can be represented in size polynomial in the number of bits needed to represent the configuration vector, the game can be represented in size polynomial in $\log n$. Thus, such a "compact" representation is able to specify games with a number of players exponential in the input size.

In this chapter, we consider the complexity of computing PSNE for symmetric games with compactly-represented utility functions. We first look at the most general setting, where the utility functions are represented as circuits whose inputs are binary representations of the configuration vector. We show that even with a fixed number of actions, the problem of deciding the existence of PSNE is NP-complete. The only exception is the case of two actions, for which we show that there always exists a PSNE and present an algorithm that identifies such an equilibrium in polynomial time.

Our main positive result is the identification of a compact representation of utility with nice computational properties —piecewise linear functions of the configuration vector. Piecewise linear functions are a natural and convenient way of representing utilities. For this setting, we present novel algorithms for finding a sample PSNE and for counting the number of PSNEs. When the number of actions is fixed, these algorithms run in polynomial time. In particular, if the total number of pieces is bounded by a polynomial of $\log n$, then we achieve an exponential improvement over the algorithm of Brandt et al. [13], which scales polynomially with n. Our techniques also yield a *polynomial-space, polynomial-delay* output-sensitive

algorithm for enumerating the set of PSNE, and an FPTAS for finding social-welfare maximizing equilibria. In addition, we provide a novel methodology for computing the average and variance of social welfare over all equilibria in a game in polynomial time, as well as a procedure for determining an empirical distribution (histogram) of the payoffs of players playing a given action a in polynomial time for a polynomial number of cells in the histogram. These latter results give insights into the distribution of utilities in a game at equilibrium, something not commonly accessible to standard methods, since in general these games possess an exponential number of equilibria in the input size.

Furthermore, we are able to extend this piecewise-linear representation to model parameterized families of symmetric games. While existing literature on equilibrium computation focused on finding equilibria in a single game, in many practical applications of game-theoretic analysis we are interested in questions about Nash equilibria of a *family* of games. For example, in mechanism design, the designer may want to choose from a family of games so that an equilibrium of the chosen game maximizes a given objective. As another example, an econometrician may be given data about agents' observed behavior and wants to estimate parameters of the underlying game. To address such problems, existing equilibrium computation approaches would be forced to solve many individual games from the family, which can be very costly. Our *parameterized symmetric games* are to our knowledge the first compact representation of parameterized families of games. Leveraging this representation, we provide a methodology for solving optimization problems over PSNE of a family of symmetric games, without having to solve each individual game in the family. These include the problem of finding parameters that ensure equilibria that are close to some observed configuration, and the problem of finding parameters that maximize the equilibrium payoff for the worst-off player.

The main challenge in constructing such polynomial-time algorithms is that the set of configurations (and, indeed, the set of PSNEs) can be exponential in the input size. Thus,

approaches based on enumerating all configurations require exponential time. Furthermore, the constraints defining PSNE are not generally convex, so the problem cannot be formulated as an integer linear program in fixed dimension. Instead, our approach encodes the set of PSNEs in a compact form that has appealing computational properties. Specifically, we make use of the *rational generating function method* for representing and computing with sets of lattice points, due to Barvinok and Woods [8]. We model configurations as integer lattice points, and formulate the set of equilibrium configurations via operations on sets of lattice points in polyhedra; the resulting set of points can be encoded as a rational generating function of polynomial size.

The results in this chapter are of similar spirit to that in Chapter 2. In that chapter we introduced rational generating function methods to the algorithmic study of games, showing that they can be used to compute pure-strategy Nash equilibria of games in which the actions are lattice points in fixed-dimensional polyhedra and the utilities are given by piecewise linear functions. These results assumed a fixed number of players and made strong restrictions on the piecewise linear functions used to represent utilities – the utilities are difference of piecewise linear convex (DPLC) functions with a "small" (fixed) number of convex pieces. In contrast, in this chapter we allow for a much more general family of piecewise utilities, including DPLC utilities with no restrictions on the number of pieces, and a non-fixed number of players. Instead we fix the number of actions. In some respects, the application of generating functions here is more natural than that of the previous chapter, since here sets of integer lattice points arise naturally as sets of configurations in the setting of symmetric games and are not artifacts of the special class of games considered.

Despite similarities, our analysis here proceeds differently than in Chapter 2. We employ different proof techniques based on disjoint set decompositions rather than relying on integer projections. A key technical insight is that when we fix the number of actions there is a fixed number of ways to deviate from a given configuration. This is the essence of how we obtain

stronger results. A more detailed description is given at the end of the current chapter, where we will also discuss implications for practical implementation of these algorithms.

Here is an overview of the content found in this chapter. Section 3.2 introduces our notation and description of the symmetric games we consider. The first are symmetric games with utilities expressed as boolean circuits. We give hardness results for deciding if pure Nash equilibria exist in games with three or more actions, and a polynomial time algorithm to find equilibria in games with two actions. The second type of symmetric game we consider are those with piecewise linear utilities. Notation and motivation are provided in this section. Section 3.3 contains the main result, that the set of pure equilibrium configurations in symmetric games can be encoded as a *short rational generating function*. An immediate corollary is that there are efficient algorithms to count and enumerate equilibria, when the number of actions m is fixed. We present two alternate methods to derive such a generating function encoding, both of which yield fresh insights into the general method.

Section 3.4 highlights how generating functions can be used to answer interesting questions about the structure of equilibria in our games. An approximation algorithm is developed for finding social welfare maximizing equilibria in the game, as well as exact algorithms to find the mean and variance of social welfare across all equilibria.

Section 3.5 details a direct generalization of our methods to the case of M-symmetric games with a fixed number of player classes M. Finally, Section 3.6 explores the situation where the number of players and utilities of each player in a symmetric game are influenced by parametric choices of a leader or designer. We are able to answer questions regarding how the designer might choose these parameters to optimize some polynomial objective, minimize error in having an equilibrium close to some target equilibrium $\tilde{\mathbf{x}}$, or maximize the worst-case payoff to the player. These results on parametric games are an appealing consequence of our method for analyzing symmetric games.

3.2 Symmetric games

A strategic game is defined by the tuple $(n, \{A_i\}_{1\leq i\leq n}, \{U_i\}_{1\leq i\leq n})$ where n is the number of players, and for each player i, A_i is her set of actions and $U_i : \prod_j A_j \to \mathbb{Z}$ is her utility function. Throughout the chapter we assume that utilities are integer-valued.

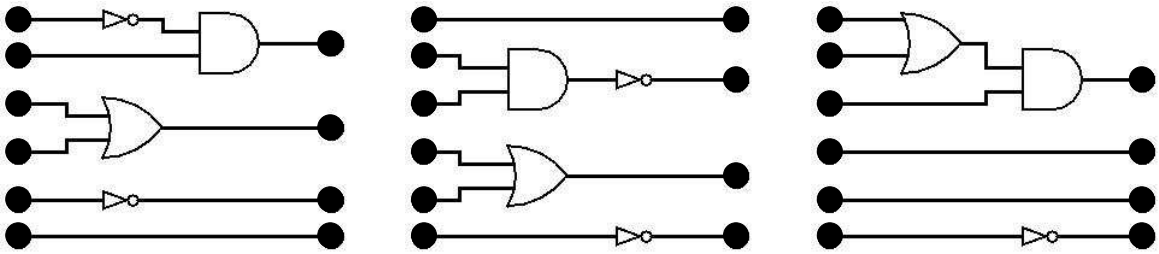
Symmetric games are a class of strategic games in which each player has an identical set of actions A and for all permutation of players $\pi : \{1, \ldots, n\} \to \{1, \ldots, n\}$, $U_i(a_1, \ldots, a_n) = U_{\pi(i)}(a_{\pi(1)}, \ldots, a_{\pi(n)})$. We consider n-player symmetric games in which the number of actions m is a fixed constant.

The outcomes of the game are sufficiently described by *configurations* of players; that is, a count of how many players take each action. A configuration is an m-dimensional vector $\mathbf{x} = (x_a : a \in A)$, where x_a is the number of players taking action a. Let D denote the set of configurations:

$$D = \left\{ \mathbf{x} \in \mathbb{Z}^m : \sum_{a\in A} x_a = n, x_a \geq 0 \text{ for all } a \in A \right\}. \tag{3.1}$$

The utility of a given player in a symmetric game depends only on the action played and the overall configuration. For each action $a \in A$, we have a function u_a defined over configurations in which at least one player takes action a. In particular, $u_a(\mathbf{x})$ is the utility of playing action a in configuration $\mathbf{x}$ (provided $x_a \geq 1$). The set of functions u_a for all $a \in A$ is sufficient for specifying the utilities of the game. For the rest of the chapter we call these u_a the utility functions and will not refer to the functions U_i.

A configuration $\mathbf{x} \in D$ is a *pure strategy Nash equilibrium configuration* (or simply a PSNE) if for all actions a and a' either no player takes action a or the utility of a player playing action a is no less than the utility he would receive from unilaterally deviating to action a'. Symbolically, let N denote the set of PSNE in a symmetric game. Then $\mathbf{x}$ is an element of N if for all $a \in A$ either $x_a = 0$ or for all $a' \in A$, $u_a(\mathbf{x}) \geq u_{a'}(\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a)$,



(a) Circuit representing $u_a(\cdot)$ (b) Circuit representing $u_b(\cdot)$ (c) Circuit representing $u_c(\cdot)$

Figure 3.1: Circuits defining a circuit symmetric game with three players and three actions.

where $\mathbf{e}_a$ is the ath unit vector with components $e_{aa} = 1$ and $e_{aa'} = 0$ for $a' \neq a$. Note that $\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a$ is the same configuration as $\mathbf{x}$ except that one player has deviated from playing action a to action a'.

3.2.1 Symmetric games with utilities as circuits

We want the utility functions to be compactly represented and efficiently computable. One approach is to model utility functions as circuits, as was done in previous work for general games [73]. In this subsection, we consider *circuit symmetric games*, a representation in which each utility function u_a is represented as a circuit whose input is a binary representation of the configuration vector $\mathbf{x}$ and output is a binary representation of the corresponding integer utility value. Since circuits can represent arbitrary integer functions of the inputs, this can represent all symmetric games with integer utilities. When utility functions can be specified as simple circuits, the representation size can be as small as $O(\log n)$.

Example 3.2.1. We describe a simple example of a circuit symmetric game for readers possibly unfamiliar with boolean circuits. Suppose there are three players $\{1, 2, 3\}$ and three actions $\{a, b, c\}$. The utility functions u_a, u_b and u_c are represented by the Boolean circuits in Figure 3.2.1. The way to interpret these circuits is as follows. The nodes to the left of each diagram are six "input" nodes and encode configurations. For example, for the configuration $(2, 1, 0)$ the input would be the triple of 2 digit binary numbers $(01, 10, 00)$ and

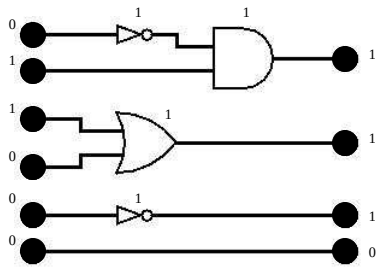


Figure 3.2: Circuit diagram representing the computation of $u_a(2,1,0) = 7$.

thus the input nodes would receive a labelling of 011000 . The nodes to the right are "output" nodes and encode in binary the utility of an input configuration. In our example, it can be seen that $u_a(2,1,0) = 7$ which has binary encoding 1110. The other nodes in the circuit diagrams are called "gates" represent logical operators. An AND gate (large arrow with a rounded head) has two incoming edges and one outgoing edge which implements the AND logical operator. An OR gate (large arrow with pointed head) has two incoming edges and one outgoing edge which implements the OR logical operator. Finally, a NOT gate (small arrow with a small circle at its head) has a single incoming edge and two outgoing edge and implements the NOT logical operator. Figure 3.2.1 demonstrates how a circuit works in computing the above-mentioned utility value $u_a(2,1,0) = 7$. By computing utilities in this manner we produce (Table 3.2.1) the explicit utilities for each configuration and action.

Configuration, $\mathbf{x}$	$u_a(\mathbf{x})$	$u_b(\mathbf{x})$	$u_c(\mathbf{x})$
$(1,1,1)$	2	15	13
$(2,1,0)$	7	8	1
$(1,2,0)$	6	15	10
$(2,0,1)$	1	8	12
$(1,0,2)$	12	3	0
$(0,2,1)$	2	14	14
$(0,1,2)$	14	2	0
$(3,0,0)$	4	11	8
$(0,3,0)$	6	14	10
$(0,0,3)$	8	6	4

Table 3.1: Utilities of each configuration and action for Example 3.2.1.

By inspection we see there are two equilibria in this game: $(0, 2, 1)$ and $(0, 3, 0)$. Note that an equilibrium in a symmetric game need not be symmetric.

For our analysis of circuit symmetric games we first consider the case with two actions, $A = \{1, 2\}$. In this case the game always has a PSNE. This follows from the fact that such a game can be formulated as a congestion game, which implies the existence of a PSNE [70]. (To see this, observe that $u_1(\mathbf{x}) = u_1(x_1, n - x_1)$ is a function of only x_1; similarly $u_2(\mathbf{x}) = u_2(n - x_2, x_2)$ is a function of only x_2.) The claim was also proven from first principles by [16].

However, even when a PSNE provably exists (or when a game is a congestion game), PSNEs can still be difficult to find. We give an alternative proof of the existence of PSNE for these games that illustrates the structure of the strategy space, and then show how this structure can be exploited for efficient computation.

Lemma 3.2.2. *Any symmetric game with two actions has a* PSNE.

Proof. Given such a symmetric game, we construct the *deviation graph*, whose vertices are the configurations $\mathbf{x} \in D$. There is a directed edge from $\mathbf{x}$ to $\mathbf{x}'$ if and only if a deviation by a single player from $\mathbf{x}$ results in $\mathbf{x}'$. Since each $\mathbf{x} = (x_1, n - x_1)$, where x_1 is the number of agents playing action 1, we can identify each configuration by its first component. Under this mapping, the set of configurations corresponds to the set of integers $\{0, \ldots, n\}$. It is straightforward to see that the only edges in the deviation graph are between adjacent integers: $i, j \in \{0, \ldots, n\}$ such that $|i - j| = 1$.

We then consider the *profitable deviation graph* (PDG), whose vertices are the same configurations and directed edges represent strictly profitable deviations. For example, if a deviation by one player in configuration $\mathbf{x}$ from action a to action $3-a$ results in configuration $\mathbf{x}'$, and furthermore if $u_{3-a}(\mathbf{x}') > u_a(\mathbf{x})$, then the PDG has an edge from $\mathbf{x}$ to $\mathbf{x}'$. Observe that the PDG is a subgraph of the deviation graph, and that if there is an edge from $\mathbf{x}$ to

$\mathbf{x}'$ in the PDG, then there cannot be an edge from $\mathbf{x}'$ to $\mathbf{x}$.

A sink of the PDG has no profitable deviations, which means that it is a PSNE. We claim that the PDG must have a sink. To see this, we can start at vertex 0 and follow the directed edges. Because the PDG is a subgraph of the deviation graph, each edge on this path must increase the vertex's index (in fact, by exactly one). Thus, the path must eventually stop at a sink. □

This proof suggests a straightforward algorithm for finding a PSNE: start at configuration 0 and follow the edges in the PDG. In fact by a similar argument any starting configuration would lead to a sink. Unfortunately this approach can take $\Omega(n)$ steps before reaching a sink, which is exponential in the representation size. Instead, we present a divide-and-conquer approach that exploits the structure of the PDG.

Theorem 3.2.3. *For circuit symmetric games with two actions, a* PSNE *can be found in polynomial time.*

Proof. Given such a game with n players, consider the configurations $\lfloor \frac{n}{2} \rfloor$ and $\lfloor \frac{n}{2} \rfloor + 1$. There are three cases:

1. If there is an edge from $\lfloor \frac{n}{2} \rfloor$ to $\lfloor \frac{n}{2} \rfloor + 1$ in the PDG, then there must exist a PSNE in the subset $\{\lfloor \frac{n}{2} \rfloor + 1, \ldots, n\}$. This is because a path from $\lfloor \frac{n}{2} \rfloor + 1$ must be increasing and eventually stop at a sink.

2. Likewise, if there is an edge from $\lfloor \frac{n}{2} \rfloor + 1$ to $\lfloor \frac{n}{2} \rfloor$, there must exist a PSNE in the subset $\{0, \ldots, \lfloor \frac{n}{2} \rfloor\}$, since a path from $\lfloor \frac{n}{2} \rfloor$ must be decreasing and stop at a sink.

3. If there is no edge between the two configurations, then there must exist a PSNE in each of the subsets $\{0, \ldots, \lfloor \frac{n}{2} \rfloor\}$ and $\{\lfloor \frac{n}{2} \rfloor + 1, \ldots, n\}$).

Our algorithm picks a subset that contains a PSNE, and then recursively bisects that subset. This process terminates at a PSNE after $O(\log n)$ iterations. For each iteration, checking

the existence of edges between two configurations requires evaluation of utility at the two configurations, which can be done in linear time for utility functions represented as circuits. Therefore the running time of this algorithm is $O(|\Gamma| \log n)$, where $|\Gamma|$ is the size of the circuits. □

Our next result shows that the problem of finding a PSNE in a circuit symmetric game becomes intractable once we go beyond two actions.

Theorem 3.2.4. *For circuit symmetric games in which the number of actions m is a fixed constant of at least three, the problem of determining the existence of* PSNE *is* NP*-complete.*

Proof. The problem is in NP because to determine whether a configuration $\mathbf{x}$ is a PSNE, there are only $O(m^2)$ possible deviations to check.

We show NP hardness by reduction from CircuitSAT. CircuitSAT is the standard NP-complete problem which asks whether there exists a binary input $\mathbf{x}$ to a circuit C which yields $C(\mathbf{x}) = 1$. Given a CircuitSAT problem instance $C(y_1, \ldots, y_m)$, we construct a circuit symmetric game with $n = 2^m - 1$ players and 3 actions $\{1, 2, 3\}$ such that the game has a PSNE if and only if there exists an satisfying assignment of $y_1, \ldots, y_m$.

Given a configuration $\mathbf{x} = (x_1, x_2, x_3)$, the utility functions $u_1(\mathbf{x})$,$u_2(\mathbf{x})$ and $u_3(\mathbf{x})$ are defined as follows:

1. If the binary representation of x_1 corresponds to a satisfying assignment for C, i.e., $C(x_1^0, \ldots, x_1^m) = 1$ where x_1^i is the ith bit of x_1, then $u_1(\mathbf{x}) = u_2(\mathbf{x}) = u_3(\mathbf{x}) = 2$.

2. Otherwise:

   (a) if $x_1 > 0, x_2 > 0, x_3 > 0$, then $u_1(\mathbf{x}) = u_2(\mathbf{x}) = 1$, $u_3(\mathbf{x}) = -2$;

   (b) if $x_1 > 0, x_2 > 0, x_3 = 0$, then $u_1(\mathbf{x}) = -1$, $u_2(\mathbf{x}) = 1$;

   (c) if $x_1 = 0, x_2 > 0, x_3 > 0$, then $u_2(\mathbf{x}) = -1$, $u_3(\mathbf{x}) = 1$;

(d) if $x_1 > 0, x_2 = 0, x_3 > 0$, then $u_1(\mathbf{x}) = 1$, $u_3(\mathbf{x}) = -1$;

(e) if $x_a = n$ for some action a, i.e., all players are playing a, then $u_a(\mathbf{x}) = 0$.

If there exists a satisfying assignment for C, then any configuration with the corresponding x_1 is a PSNE because each player receives the maximum utility of the game. If there does not exist a satisfying assignment, then the game's utilities are defined by condition 2. We claim that this subgame under condition 2 does not have a PSNE. Intuitively, the game can be thought of as a generalization of the 2-player Rock-Paper-Scissors game. Formally, given a configuration of case 2a, a deviation from action 3 (with utility -2) to 1 or 2 is profitable. Given a configuration of case 2b, a profitable deviation is from action 1 (utility -1) to 2 (utility 1 if the resulting configuration is of case 2b, utility 0 if the resulting configuration is of case 2e). Similarly, given a configuration of case 2c, a profitable deviation is from action 2 to 3; and given a configuration of case 2d, a profitable deviation is from action 3 to 1. Given a configuration of case 2e with e.g. $x_1 = n$, a profitable deviation is to action 2, resulting in a configuration of case 2b. Therefore all configurations have profitable deviations, thus the subgame does not have a PSNE.

Finally, we observe that the utility functions described above can be formulated as circuits of the binary representation of $\mathbf{x}$. The size of the circuit symmetric game is linear in the size of the given CircuitSAT problem instance, and these utility functions can be constructed in polynomial time. This concludes the reduction proof. □

3.2.2 Symmetric games with piecewise linear utilities

The hardness result of Theorem 3.2.4 suggests that the circuit symmetric game representation is too general to be computationally useful in the worst case: it is powerful enough to encode NP-hard problems. Thus, computation of PSNE is intractable even if the representation is compact.

In this subsection and the rest of the chapter, we consider utility functions represented as piecewise-linear functions. Piecewise-linear functions have been widely used as an approximation of arbitrary continuous functions. A recent example of their use in an economic context is [34], which considered piecewise-linear utilities in the computation of market equilibria in the Arrow-Debreu model, and presented a polynomial-time algorithm when utilities are piecewise-linear concave functions and the number of goods is constant.

Also, given an arbitrary utility function, it can be described exactly by a piecewise linear function, although the number of pieces required may be $\Theta(n^m)$ in general, in which case it is no longer compact in the sense we defined at the beginning of the chapter.

Even if the utility functions cannot be exactly represented as piecewise-linear functions with a small number of pieces, piecewise-linear functions can still be useful as approximations of such utility functions. If for all configurations $\mathbf{x}$ and all actions a the difference between $u_a(\mathbf{x})$ and its approximation $\hat{u}_a(\mathbf{x})$ is at most ϵ, then by a standard argument any Nash equilibrium of the approximate game is a 2ϵ-Nash equilibrium of the original game. An ϵ-Nash equilibrium is an action profile in which each player can gain at most ϵ by deviating from her current strategy. More precisely, a configuration $\mathbf{x} \in D$ is an ϵ-pure strategy Nash equilibrium configuration (ϵ-PSNE) for all $a \in A$ either $x_a = 0$ or for all $a' \in A$, $u_a(\mathbf{x}) \geq u_{a'}(\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a) - \epsilon$.

We consider two approaches to defining piecewise linear utilities. The first is maybe more standard, where the utilities are described by explicitly specifying a polytopal subdivision of the domain and an affine function for each cell. The domain for our utility functions is the set of configurations D. Thus we define a utility function by specifying a set of polytopes that induces a partition of the integer points D, and an affine function for each cell. We call utility functions specified in this way PWL utilities (short for piecewise linear).

Formally, for each action $a \in A$, the piecewise linear (PWL) utility function $u_a(\mathbf{x})$ is given as follows. There is a finite set of polytopes $\{P_{aj}\}_{j \in J_a}$ with index set J_a where the set

of configurations D is partitioned by the integer points in these polytopes. In other words,

$$D = \biguplus_{j \in J_a} (P_{aj} \cap \mathbb{Z}^m),$$

where the notation $\biguplus$ signifies disjoint union. Each polytope $P_{aj} = \{\mathbf{x} \in \mathbb{R}^m : M_{aj}\mathbf{x} \leq \mathbf{b}_{aj}\}$ is given by an integer matrix M_{aj} and integer right-hand side vector $\mathbf{b}_{aj}$. Over each cell $P_{aj} \cap \mathbb{Z}^m$ there is an affine function $f_{aj}(\mathbf{x}) = \boldsymbol{\alpha}_{aj} \cdot \mathbf{x} + \beta_{aj}$ with $\boldsymbol{\alpha}_{aj} \in \mathbb{Z}^m$ and $\beta_{aj} \in \mathbb{Z}$, such that

$$u_a(\mathbf{x}) = f_{aj}(\mathbf{x}) \text{ for } \mathbf{x} \in P_{aj} \cap \mathbb{Z}^m. \tag{3.2}$$

Thus, the piecewise linear utility function $u_a(\mathbf{x})$ is input as the binary encoding of M_{aj}, $\mathbf{b}_{aj}$, $\boldsymbol{\alpha}_{aj}$ and β_{aj} for each $j \in J_a$.

Example 3.2.5. To illustrate this definition we consider a simple example of a symmetric game with piecewise linear utilities. We consider a game with 10 players and 3 actions a, b and c. The set D is the integer points in the tenth dilate of the unit simplex; that is, the nonnegative integer vectors in $\mathbb{Z}^3$ whose components sum to 10. We now describe the piecewise linear utility functions.

The utility $u_a(x_1, x_2, x_3)$ associated with action a is defined on the partition of $D = P_{a1} \cap \mathbb{Z}^3 \uplus P_{a2} \cap \mathbb{Z}^3$ given by the two polytopes

$$P_{a1} = \{\mathbf{x} \in \mathbb{R}^3 : \sum_{i=1}^{3} x_i = 1,\ x_1 \leq 8,\ x_i \geq 0,\ \text{for } i = 1, 2, 3\}$$

and

$$P_{a2} = \{\mathbf{x} \in \mathbb{R}^3 : \sum_{i=1}^{3} x_i = 1,\ x_1 \geq 9,\ x_i \geq 0,\ \text{for } i = 1, 2, 3\}.$$

The affine piece above P_{a1} is $f_{a1}(x_1, x_2, x_3) = x_1 + x_2 + x_3$, and the affine piece above P_{a2} is

$f_{a2}(x_1, x_2, x_3) = 2x_1 - x_2 - x_3$. That is,

$$u_a(x_1, x_2, x_3) = \begin{cases} x_1 + x_2 + x_3, & (x_1, x_2, x_3) \in P_{a1} \cap \mathbb{Z}^3 \\ 2x_1 - x_2 - x_3, & (x_1, x_2, x_3) \in P_{a2} \cap \mathbb{Z}^3 \end{cases}$$

We similarly define u_b as follows:

$$u_b(x_1, x_2, x_3) = \begin{cases} -2x_1 - x_2 + x_3, & (x_1, x_2, x_3) \in P_{b1} \cap \mathbb{Z}^3 \\ -x_1 + x_2 + 2x_3, & (x_1, x_2, x_3) \in P_{b2} \cap \mathbb{Z}^3 \end{cases}$$

where

$$P_{b1} = \{\mathbf{x} \in \mathbb{R}^3 : \sum_{i=1}^{3} x_i = 1,\ x_2 \leq 4,\ x_i \geq 0,\ \text{for } i = 1, 2, 3\}$$

and

$$P_{b2} = \{\mathbf{x} \in \mathbb{R}^3 : \sum_{i=1}^{3} x_i = 1,\ x_2 \geq 5,\ x_i \geq 0,\ \text{for } i = 1, 2, 3\}.$$

Finally, u_c is defined by

$$u_c(x_1, x_2, x_3) = \begin{cases} x_1 - x_2 + x_3, & (x_1, x_2, x_3) \in P_{c1} \cap \mathbb{Z}^3 \\ -x_1 + 2x_2 - x_3, & (x_1, x_2, x_3) \in P_{c2} \cap \mathbb{Z}^3 \end{cases}$$

where

$$P_{c1} = \{\mathbf{x} \in \mathbb{R}^3 : \sum_{i=1}^{3} x_i = 1,\ x_3 \leq 6,\ x_i \geq 0,\ \text{for } i = 1, 2, 3\}$$

and

$$P_{c2} = \{\mathbf{x} \in \mathbb{R}^3 : \sum_{i=1}^{3} x_i = 1,\ x_3 \geq 7,\ x_i \geq 0,\ \text{for } i = 1, 2, 3\}.$$

Note that the input size of this symmetric game is roughly the econding size of 24 integers; that is, the integral data defining the affine pieces and polytopes. However, the encoding size of explicitly representing the utility for each pair of action and configuration is the encoding

size of $3\binom{10+3-1}{3-1} = 3\binom{12}{2} = 198$ integers. This illustrates the potential compactness inherent in a piecewise linear structure.

Secondly, we consider an implicit form of piecewise linear representation called *difference of piecewise linear convex* (DPLC) utilities. The input here is a pair of sets of affine functions with integer coefficients $\{u_{ak}\}_{k \in K_a}$ and $\{v_{al}\}_{l \in L_a}$ such that

$$u_a(\mathbf{x}) = \max_{k \in K_a} u_{ak}(\mathbf{x}) - \max_{l \in L_a} v_{al}(\mathbf{x}). \tag{3.3}$$

Where convenient we label the elements of K_a by $\{1, \ldots, |K_a|\}$, and similarly for L_a. Without loss of generality we assume that the utility functions are all positive over the domain D.

Theoretically our two representations of piecewise linear objectives are equivalent, as stated in the following proposition:

Proposition 3.2.6. *A utility function over D can be represented as an explicit piecewise linear utility function of the form* (3.2) *if and only if it can be represented as a* DPLC *utility function of the form* (3.3)*.*

Proof. We first prove the if-direction. Given a DPLC function of the form (3.3) we consider partitioning the set of configurations D into regions where the objectives are linear. For each action $a \in A$ partition D as follows

$$D = \left(\biguplus_{(k,l) \in K_a \times L_a} D^a_{k,l} \right)$$

where

$$D^a_{k,l} = \left\{ \mathbf{x} \in D : x_a \geq 1, \quad \begin{array}{llll} u_{ak}(\mathbf{x}) \geq u_{aj}(\mathbf{x}), & j > k, & u_{ak}(\mathbf{x}) \geq u_{aj}(\mathbf{x}) + 1, & j < k \\ v_{al}(\mathbf{x}) \geq v_{aj}(\mathbf{x}), & j > l, & v_{al}(\mathbf{x}) \geq v_{aj}(\mathbf{x}) + 1, & j < l \end{array} \right\}.$$

is the set of elements in D where $\max_{j \in K_a} u_{aj}(\mathbf{x}) = u_{ak}(\mathbf{x})$ and $\max_{j \in L_a} v_{aj}(\mathbf{x}) = v_{al}(\mathbf{x})$ breaking ties to make the sets disjoint (see Lemma 2.3.1 for a similar discussion). Now if we let $J_a = K_a \times L_a$ then our explicit representation of u will have $P_{aj} = D^a_{k,l}$ and $f_{aj}(\mathbf{x}) = u_{ak}(\mathbf{x}) - v_{al}(\mathbf{x})$ for $j = (k, l) \in J_a$.

Conversely, given an explicitly represented piecewise linear function, a result by Zalgaller (Theorem 4.2 in [86]) shows that it can be represented by a DPLC function. In his proof Zalgaller shows that our explicitly represented piecewise-linear function u_a can be expressed as the difference $u_a = g - h$ where $h = \sum_{i=1}^{M} h_i$ with each h_i a piecewise linear function with two pieces and $g = u_a + h$, which is also shown to be a piecewise linear convex function. The number M of functions h_i in the sum defining h depends on the nature of the polytopal division $\{P_{aj}\}_{j \in J_a}$ of D and is bounded above by the total number of facets in the polytopes P_{aj}. The proof in [86] is not explicit enough for us to easily show how to convert g and h into "maximum of affine functions" form. Indeed, there is some flexibility in how to choose the h_i. It is clear, however, that such a representation exists as every piecewise linear convex function has a "maximum of affine functions" representation. □

In practice, the conversion from an explicitly represented piecewise linear function into a DPLC function seems of little value, but the opposite conversion, which is given explicitly in the above proposition may be of interest depending on the context. Although in theory these two representations of utilities are equivalent, from the point of view of representation sizes they can be quite different. The benefit of the DPLC representation is that there is no need to explicitly represent the polytopal subdivision of D, which may yield savings in terms of input size. Although the explicit representation of a piecewise linear utility is typical, there are settings where a DPLC formulation is more natural. For instance, it is common in economic applications for cost functions to be expressed as piecewise linear concave functions in "minimum of affine functions" form. The affine functions represent production processes with different start-up fixed costs (intercepts) and per-unit costs of production (slopes). A

manufacturer then produces using the production process with minimum total cost. Finally, we observe that a piecewise linear utility function in PWL or DPLC form can be represented as a circuit.[4]

3.3 Main result

This chapter's main positive result follows.

Theorem 3.3.1 (intuitive version)**.** *Consider a symmetric game with* PWL *or* DPLC *utilities given by a binary encoding of the number of players and of the utility functions. Then, when the number of actions m is fixed, there exists*

(i) a polynomial-time algorithm to compute the number of pure Nash equilibrium configurations;

(ii) a polynomial-time algorithm to find a sample pure strategy Nash equilibrium, when at least one exists; and

(iii) a polynomial-space, polynomial-delay enumeration algorithm to enumerate all the pure Nash equilibrium configurations of the game.

The proof of this theorem draws on the theory of rational generating functions for encoding lattice points sets in polyhedra. The methodology is similar to Chapter 2 with some important differences which are discussed in Section 3.7. The proof is derived in our two cases in Theorems 3.3.2 and 3.3.5.

[4]Addition and multiplication can be carried out by circuits. To determine which piece a configuration is in, we can go through each piece and test the inequalities. The size of the resulting circuit is polynomial in the number of bits needed to describe the piecewise linear function.

3.3.1 PWL case: A decomposition-based encoding

To start we will consider PWL utilities of the form (3.2). To establish the result we take advantage of the polyhedral subdivisions of D which define the utility functions and view this in terms of boolean operations involving lattice points of polyhedra.

We now state and prove our main theorem formally for the PWL case:

Theorem 3.3.2 (PWL version)**.** *Consider a symmetric game with* PWL *utilities given by the following input:*

(I_1) *the binary encoding of the number n of players;*

(I_2) *for each $a \in A$, the utility function $u_a(\mathbf{x})$ represented as the binary encoding of M_{aj}, $\mathbf{b}_{aj}$, $\boldsymbol{\alpha}_{aj}$ and β_{aj} for each $j \in J_a$.*

Then, when the number of actions m is fixed, there exists

(i) a polynomial-time algorithm to compute the number of pure Nash equilibrium configurations;

(ii) a polynomial-time algorithm to find a sample pure strategy Nash equilibrium, when at least one exists; and

(iii) a polynomial-space, polynomial-delay enumeration algorithm to enumerate all the pure Nash equilibrium configurations of the game.

Proof. We first show that N, the set of PSNE, can be encoded as a short rational generating function. Let N denote the set of PSNE. A difficulty in applying generating functions to encoding N is the nonlinearity of the objectives u_a. However, since these objectives are piecewise linear, we simply consider the partitions of D into regions in which the objectives *are* linear. We use these partitions of D (and hence of N) to express N as a Boolean combination of polytopal lattice point sets, and thus will ultimately be able to apply Lemma 1.4.4.

The overall idea is to define subsets of configurations that have strictly profitable deviations, then remove these subsets from D, leaving only the set of PSNE.

Define the *deviation set* $Dev(a, a', j, j')$ as the set of configurations $\mathbf{x}$ in which a player currently playing action a lying in region P_{aj} has a *strictly* profitable deviation to playing action a', thereby yielding a new configuration $\mathbf{x}' \in P_{a'j'}$. Such a profitable deviation will exist whenever $f_{aj}(\mathbf{x}) < f_{a'j'}(\mathbf{x}')$. Since the affine functions have integer coefficients, we can rewrite this condition as $f_{aj}(\mathbf{x}) \leq f_{a'j'}(\mathbf{x}') - 1$ and thereby avoid strict inequalities. Putting this together, we define the deviation set as

$$Dev(a, a', j, j') = \left\{ \begin{array}{l} \mathbf{x} \in D : x_a \geq 1, \mathbf{x} \in P_{aj}, \\ \mathbf{x}' = \mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a \in P_{a'j'} \\ f_{aj}(\mathbf{x}) \leq f_{a'j'}(\mathbf{x}') - 1 \end{array} \right\}. \tag{3.4}$$

Now we can use (3.4) to describe N. We obtain

$$N = D \setminus \bigcup_{a,a'} \biguplus_{j} \biguplus_{j'} Dev(a, a', j, j'), \tag{3.5}$$

where the first union is over all $a, a' \in A$, the second union is over $j \in J_a$, and the third union is over $j' \in J_{a'}$. This identity (ignoring for now our claim that the second two unions are disjoint) can be verified as follows. Suppose configuration $\mathbf{x} \in D$ *does not* lie in the right-hand side of (3.5). This implies that $\mathbf{x}$ lies in some deviation set $D(a, a', j, j')$ for some $a, a' \in A$ and $(j, j') \in J_a \times J_{a'}$ and hence there is a profitable unilateral deviation away from $\mathbf{x}$, implying that $\mathbf{x}$ is not in N. Conversely, suppose $\mathbf{x} \in D$ is not in N. This implies that there exists a profitable unilateral deviation, say from playing action a to a'. This implies $u_a(\mathbf{x}) < u_{a'}(\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a)$. Now, $\mathbf{x}$ and $\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a$ lie in cells P_{aj} for some $j \in J_a$ and $P_{a'j'}$ for some $j' \in J_{a'}$ respectively. The condition on the utilities then implies that $f_{aj}(\mathbf{x}) \leq f_{a'j'}(\mathbf{x}') - 1$. It follows that $\mathbf{x}$ is in the deviation set $Dev(a, a', j, j')$ and thus not

contained in the righthand side of (3.5).

Now we substantiate our claims that the second two unions are disjoint. The union indexed by j is disjoint because the sets $\{P_{aj}\}_{j \in J_a}$ form a partition of D, and $Dev(a, a', j, j') \subseteq P_{aj}$. To see that the union indexed by j' is disjoint, consider an arbitrary element $\mathbf{x}$ of $Dev(a, a', j, j')$. This implies that $\mathbf{x}' = \mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a \in P_{a'j'}$. Because $\{P_{a'j'}\}_{j' \in J_{a'}}$ are disjoint sets, for all $j'' \in J_{a'} \setminus \{j'\}$ we have $\mathbf{x}' \notin P_{a'j''}$ and thus $\mathbf{x} \notin Dev(a, a', j, j'')$. Therefore $Dev(a, a', j, j')$ is disjoint from $Dev(a, a', j, j'')$ for any $j', j'' \in J_{a'}, j' \neq j''$.

We took particular care in describing which unions in our expression for N are disjoint and which are not. This is because the second and third unions are not of fixed length as would be required for the application of the Boolean Operations Lemma (Lemma 1.4.4). However, since the unions are disjoint we can use simple addition of generating functions to handle this part of the overall expression of N. To make this precise, note that each of the $Dev(a, a', j, j')$ terms are polytopal lattice point sets and thus admit Barvinok encodings $g(Dev(a, a', j, j'), \boldsymbol{\xi})$ that can be computed in polynomial time by Lemma 1.4.3.

For each $a, a' \in A$ define

$$Dev(a, a') = \biguplus_{j} \biguplus_{j'} Dev(a, a', j, j'). \tag{3.6}$$

$Dev(a, a')$ is a disjoint union, and so by Lemma 1.4.5 also admits a Barvinok encoding

$$g(Dev(a, a'), \boldsymbol{\xi}) = \sum_{j} \sum_{j'} g(Dev(a, a', j, j'), \boldsymbol{\xi}).$$

We can use (3.6) to rewrite (3.5) as $N = D \setminus \bigcup_{a,a' \in A} Dev(a, a')$. Since D and each of the sets $Dev(a, a')$ have Barvinok encodings, Lemma 1.4.4 tells us that we can also derive such an encoding for N. This Boolean combination of sets describing N is of constant length since the number of actions m is fixed.

Now that we have shown that N can be encoded as a short rational generating function, (i)-(iii) follow by applying Lemmas 1.4.8 and 1.4.9. Given that we can compute $g(N, \boldsymbol{\xi})$ in polynomial time, we can compute its cardinality in polynomial time by applying Lemma 1.4.8. This establishes (i). Applying Lemma 1.4.9 (noting the bound $N \subseteq [0, n]^m$) we need only wait polynomial time to output the first element of N, establishing (ii). The enumeration scheme (iii) derives from Lemma 1.4.9 directly. □

Of course this result trivially answers the problem of deciding if pure Nash equilibria exist in symmetric games with PWL utilities, simply count the number of equilibria by the above algorithm and observe if the number is nonzero. This can be done in polynomial time when the number of actions m is fixed. When we contrast this result with that of Brandt et al. [13] we see that the input of a utility value for each action and configuration has been replaced with an input of the binary encoding of each piece of the PWL function which describes the utilities. In particular, since the input is the number n and its encoding size is $\log n$ the number of players is *exponential* in the input size if the other input data, including the number of pieces, scales with $\log n$. By contrast the algorithm of Brandt et al. [13] scales with n.

3.3.2 DPLC case: Encoding extended equilibria

In the case of DPLC utilities we could apply Proposition 3.2.6 to convert DPLC utility functions into PWL utility functions and then apply Theorem 3.3.2 to the to yield an encoding of N. Despite this, we present an alternate approach to encoding N, which follows a different argument and takes as input the DPLC description of utilities given in (3.3), which is more compact than its PWL equivalent – it has fewer pieces to encode and no need for describing a polytopal subdivision of D directly. In addition, we believe the following approach to encoding N is of interest in its own and is thus included. The idea of the encoding

is to introduce auxiliary variables $\mathbf{y} \in \mathbb{Z}^{A\times A}$ and $\mathbf{z} \in \mathbb{Z}^{A\times A}$ which capture information about the utilities. In particular, we let $y_{aa'}$ represent the contribution to the "convex" pieces $u_{a'k}$ to the utility function $u_{a'}$ when there is a deviation from the current configuration $\mathbf{x}$ to one where one player switches from playing a to a'. The variable $z_{aa'}$ is similar but corresponds to the "concave" pieces $v_{a'l}$. This idea is similar to the extended game approach found in the previous chapter, so we will be somewhat more terse in the presentation here. The relationship we want to establish between these new variables, the configurations $\mathbf{x}$ and the utilities are defined by the following set of *extended equilibrium configurations* $\hat{N}$:

$$\hat{N} = \left\{ (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in D \times \mathbb{Z}^{A\times A} \times \mathbb{Z}^{A\times A} : \forall a, a' \in A : \begin{array}{ll} y_{aa'} = \max_{k\in K_{a'}} u_{a'k}(\mathbf{x} + \mathbf{e}'_a - \mathbf{e}_a), & \\ z_{aa'} = \max_{l\in L_{a'}} v_{a'l}(\mathbf{x} + \mathbf{e}'_a - \mathbf{e}_a), & \\ y_{aa} - z_{aa} \geq y_{aa'} - z_{aa'} & \text{when } x_a \geq 1 \end{array} \right\}. \tag{3.7}$$

Our goal is to encode $\hat{N}$ as a rational generating function and leverage this to understand N. The following propositions reveals an important relationship between $\hat{N}$ and N.

Proposition 3.3.3. *The exists a bijection between the sets N and $\hat{N}$, where N is the set of pure Nash equilibrium configurations and $\hat{N}$ is the set of extended equilibrium configurations defined in* (3.7)*.*

Proof. Consider the mapping $\varphi : N \longrightarrow \hat{N}$ defined by $\mathbf{x} \longmapsto (\mathbf{x}, \mathbf{y}, \mathbf{z})$ where for all $a, a' \in A$, $y_{aa'} = \max_{k\in K_{a'}} u_{a'k}(\mathbf{x} + \mathbf{e}'_a - \mathbf{e}_a)$ and $z_{aa'} = \max_{l\in L_{a'}} v_{a'k}(\mathbf{x} + \mathbf{e}'_a - \mathbf{e}_a)$. In particular the mapping yields: $u_a(\mathbf{x}) = y_{aa} - z_{aa}$ and more generally, $u_{a'}(x + \mathbf{e}_{a'} - \mathbf{e}_a) = y_{aa'} - z_{aa'}$.

We first claim that φ is well-defined. If $\mathbf{x} \in N$ then for all $a \in A$ either $x_a = 0$ or for all $a' \in A$, $u_a(\mathbf{x}) \geq u_{a'}(\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a)$. Thus, this implies that if $x_a \geq 1$ then $y_{aa} - z_{aa} \geq y_{aa'} - z_{aa'}$ and hence $(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \hat{N}$. The fact that φ can be verified by similar reasoning to that found in Lemma 2.3.5. □

Theorem 3.3.4. *Consider a symmetric game with* DPLC *utilities given by* (3.3) *with input, in binary encoding:*

(I_1) *the number n of players;*

(I_2) *for each $a \in A$, nonnegative integers $|K_a|$ and $|L_a|$, and for all integers $k \in K_a$ and $l \in L_a$, integer vectors $\boldsymbol{\alpha}_{ik} \in \mathbb{Z}^m$, $\boldsymbol{\gamma}_{il} \in \mathbb{Z}^m$ and integers β_{ak}, δ_{ak} defining the affine functions u_{ak} and v_{al} by $u_{ak}(\mathbf{x}) = \boldsymbol{\alpha}_{ik} \cdot \mathbf{x} + \beta_{ik}$ and $v_{al}(\mathbf{x}) = \boldsymbol{\gamma}_{al} \cdot \mathbf{x} + \delta_{il}$ for all $\mathbf{x} \in D$.*

Then there exists a polynomial time algorithm which computes a short rational generating function encoding of $\hat{N}$ when the number of actions m is fixed.

Proof. Let

$$\hat{D} = \left\{ (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in D \times \mathbb{Z}^{A \times A} \times \mathbb{Z}^{A \times A} : \begin{array}{c} \forall a, a' \in A : \quad \forall k \in K_{a'},\ u_{a'k}(\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a) \leq\ y_{aa'} \leq U'_a, \\ \forall l \in L_{a'},\ v_{a'k}(\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a) \leq z_{aa'} \leq V'_a, \end{array} \right\}.$$

where U'_a and V'_a are upper bounds on the value of the utilities, as defined below. The reason for including these bounds is in anticipation of applying Barvinok's algorithm for encoding the set $\hat{D}$ as a short rational generating function, which applies only to polytopes.

We define the bound $U_{a'}$ for $y_{aa'}$ as follows, $V_{a'}$ is defined analogously. Let $U_{a'k}$ denote the optimal value of the following linear program:

$$\max\{u_{a'k}(\mathbf{x}) : \mathbf{x} \in \mathrm{conv}(D)\}.$$

The feasible set $\mathrm{conv}(D)$ is the convex hull of the set of feasible configurations D. Thus, the value $U_{a'k}$ can be found in polynomial time (say by the ellipsoid method) and is of size polynomial in the binary encoding of the input. Note that $U_{a'k}$ is an upper bound on the corresponding integer program: $\max\{u_{a'k}(\mathbf{x}) : \mathbf{x} \in D\}$.

Let $U_{a'} = \max_k U_{a'k}$. Again, this can be computed in polynomial time, since K_a is part of the input. This bound does not overly restrict the choice of $y_{aa'}$ since we have the relation:

$$y_{aa'} \leq \max_k u_{a'k}(\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a) \leq \max_k \max_{\mathbf{x}} u_{a'k}(\mathbf{x}) = \max_k U_{a'k} = U_{a'}.$$

Having described $\hat{D}$, the next step is to express $\hat{N}$ as a Boolean combination of polytopal lattice point sets. Here is one possible way of expressing $\hat{N}$ in this form:

$$\hat{N} = \hat{D} \setminus \left(\bigcup_{a,a' \in A} Y^{>}_{a,a'} \right) \setminus \left(\bigcup_{a,a' \in A} Z^{>}_{a,a'} \right) \setminus \left(\bigcup_{a,a' \in A} W^{>}_{a,a'} \right)$$

where

$$Y_{a,a'} = \{(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \hat{D} : \forall k \in K_{a'} : y_{aa'} \geq u_{a'k}(\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a) + 1\},$$

$$Z_{a,a'} = \{(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \hat{D} : \forall l \in L_{a'} : z_{aa'} \geq v_{a'k}(\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a) + 1\},$$

and

$$W_{a,a'} = \{(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \hat{D} : x_a \geq 1, y_{aa} - z_{aa} \leq y_{aa'} - z_{aa'} - 1\}.$$

We can view $Y_{a,a'}$ as the set of $(\mathbf{x}, \mathbf{y}, \mathbf{z})$ where for $a, a' \in A$ the values of $y_{aa'}$ is too large for the set $\hat{N}$. We can similarly interpret the sets $Z_{a,a'}$ and $W_{a,a'}$. It is straightforward to verify that this is indeed a valid description of $\hat{N}$.

Each of $Y_{a,a'}$, $Z_{a,a'}$ and $W_{a,a'}$ are polytopal lattice points sets and therefore can be encoded by short rational generating functions in polynomial time. The Boolean expression for $\hat{N}$ thus consists of a fixed number of set operations over sets with short rational generating function encodings and thus $\hat{N}$ itself can be encoded by a short rational generating function in polynomial time, by the Boolean Operations Lemma (Lemma 1.4.4). □

Corollary 3.3.5. *Consider an symmetric integer programming game with* DPLC *payoffs of the form 3.3 with input as in the previous theorem. Then, the set N of pure Nash equilibrium*

configurations has a short rational generating function encoding, which can be computed in polynomial time when the number of actions m is fixed.

Proof. With an encoding of $\hat{N}$ we get an encoding of N by simply noting the bijection in Proposition 3.3.3 and applying Remark 1.4.6. □

As in the PWL case we can use this encoding to count and enumerate equilibria. In the remainder of this chapter we explore extensions and applications of the encoding results found in this section for both the PWL and DPLC utility function setting. A proof in one setting leads naturally to an analogous ideas for a proof in the other. The first of these results (Theorem 3.4.3), however, has complete proofs for both settings to give the reader the taste for the arguments. Later on we only establish results carefully for the PWL case, where we hope the parallel result to the DPLC case is clear.

3.4 Exploring the structure and distribution of equilibrium configurations

This section examines how, once a generating function encoding of the set of pure Nash equilibria has been attained, this encoding can be used to further analyze the structure of equilibria. In so doing, we demonstrate the power of the rational generating functions in allowing for efficient computation.

First we derive algorithms to compute approximate social welfare maximizing equilibria. Next we demonstrate how generating functions can be used to learn about the distributions of welfare amongst the set of equilibria, in terms of calculating the mean and variance of social welfare over the set N. Lastly, we explore the distributions of equilibrium payoffs by deriving histograms, in addition to mean and variance, for the distribution of payoffs earned from playing a particular action a in equilibrium. All of this gives us further insight into the

structure and characteristics of alternate equilibria in symmetric games.

We first give basic extensions of Lemma 1.4.10 and Lemma 1.4.11 to the case of optimizing piecewise-linear and piecewise-polynomial functions over a lattice point set described by its Barvinok encoding. They will be used here and in Section 3.6.

Lemma 3.4.1 (**Piecewise-Linear Optimization**). *Let $S \in \mathbb{Z}^m$ be a lattice point set with Barvinok encoding $g(S, \boldsymbol{\xi})$ and fix m (the dimension) and ℓ (the maximum number of binomials in the denominators of the rational terms in $g(S, \boldsymbol{\xi})$). Then there exists a polynomial-time algorithm for the following problem. Given as* input*, in binary encoding,*

(I_1) *a number $M \in \mathbb{Z}_+$;*

(I_2) *the Barvinok encoding of a finite set $S \subseteq [-M, M]^m$;*

(I_3) *for each $j \in J$ an inequality description $(M_j, \mathbf{b}_j)$ of rational polytope $P_j = \{\mathbf{x} \in \mathbb{R}^{d_j} : M_j\mathbf{x} \leq \mathbf{b}_j\} \subset [-M, M]^k$ such that $S = \biguplus_{j \in J} S \cap P_j$;*

(I_4) *For each $j \in J$ vector $\boldsymbol{\alpha}_j \in \mathbb{Z}^m$ and scalar $\beta_j \in \mathbb{Z}$ defining affine functions $a_j(\mathbf{x}) = \boldsymbol{\alpha}_j \cdot \mathbf{x} + \beta_j$ which compose the piecewise linear function*

$$a(\mathbf{x}) = \{a_j(\mathbf{x}) : \mathbf{x} \in P_j\};$$

output*, in binary encoding,*

(O_1) *an optimal solution $\mathbf{x}^*$ of the optimization problem* $\max\{a(\mathbf{x}) : \mathbf{x} \in S\}$.

Proof. First for each $j \in J$ compute a Barvinok encoding of each $S \cap P_j$ using the Boolean Operations Lemma (Lemma 1.4.4). Then observe that

$$\max\{a(\mathbf{x}) : \mathbf{x} \in S\} = \max_{j \in J} \max\{\boldsymbol{\alpha}_j\mathbf{x} + \beta_j : \mathbf{x} \in S \cap P_j\}$$

Each of the inner optimization problems can be solved in polynomial time using the Linear Optimization Lemma (Lemma 1.4.10). □

We can also optimize over piecewise-polynomial functions, provided each polynomial is nonnegative over its domain.

Lemma 3.4.2 (**FPTAS for maximizing piecewise polynomials**). *Let $S \in \mathbb{Z}^m$ be a lattice point set with Barvinok encoding $g(S, \boldsymbol{\xi})$ and fix m (the dimension) and ℓ (the maximum number of binomials in the denominators of the rational terms in $g(S, \boldsymbol{\xi})$). Then there exists a polynomial-time algorithm for the following problem. Given as* input *an*

(I_1) *two vectors $\mathbf{x}_\mathrm{L}$, $\mathbf{x}_\mathrm{U} \in \mathbb{Z}^m$,*

(I_2) *the Barvinok encoding of a finite set $S \subseteq \mathbb{Z}^m$ of lattice points that is contained in the box $B = \{\, \mathbf{x} : \mathbf{x}_\mathrm{L} \leq \mathbf{x} \leq \mathbf{x}_\mathrm{U} \,\}$,*

(I_3) *for each $j \in J$ an inequality description $(M_j, \mathbf{b}_j)$ of rational polytope $P_j = \{\mathbf{x} \in \mathbb{R}^{d_j} : M_j\mathbf{x} \leq \mathbf{b}_j\} \subset B$ such that $S = \biguplus_{j \in J} S \cap P_j$;*

(I_4) *for each $j \in J$ a list of coefficients $f_{ij} \in \mathbb{Q}$, encoded in binary encoding, and exponent vectors $\boldsymbol{\alpha}_{ij} \in \mathbb{Z}^m_+$, encoded in unary encoding, representing piecewise polynomial $f(\mathbf{x}) = \{f_j(\mathbf{x}) : \mathbf{x} \in P_j\}$, where*

$$f_j = \sum_i f_{ij} \mathbf{x}^{\boldsymbol{\alpha}_{ij}}$$

and each f_j is non-negative on S;

(I_5) *a positive rational number $1/\epsilon$ encoded in unary encoding,*

output, *in binary encoding,*

(O_1) *a point* $\mathbf{x}_\epsilon \in S$ *that satisfies*

$$f(\mathbf{x}_\epsilon) \geq (1-\epsilon)f^* \quad \textit{where} \quad f^* = \max_{\mathbf{x}\in S} f(\mathbf{x}).$$

Proof. Let $\epsilon > 0$ be given. First for each $j \in J$ compute a Barvinok encoding of each $S \cap P_j$ using the Boolean Operations Lemma (Lemma 1.4.4). For all $j \in J$ apply the algorithm for polynomial optimization from Lemma 1.4.11 to find an ϵ-optimal solution $\mathbf{x}_\epsilon^j$ for each optimization problem: $\max\{f_j(\mathbf{x}) : \mathbf{x} \in N \cap P_j\}$. Then set $\mathbf{x}_\epsilon$ to be an $\mathbf{x}_\epsilon^j$ that satisfies

$$f_j(\mathbf{x}_\epsilon^j) \geq f_k(\mathbf{x}_\epsilon^k) \quad \forall k \in J.$$

We claim $\mathbf{x}_\epsilon$ satisfies the required condition $f(\mathbf{x}_\epsilon) \geq (1-\epsilon)f^*$. An optimal solution $\mathbf{x}^* \in S$ (i.e., $f(\mathbf{x}^*) = f^*$) must lie in a set $S \cap P_j$ for some $j \in J$. Hence $f^* = f_j(\mathbf{x}^*)$. Now, by the definition of $\mathbf{x}_\epsilon^j$,

$$f_j(\mathbf{x}_\epsilon^j) \geq (1-\epsilon)f_j(\mathbf{x}) \quad \forall \mathbf{x} \in N \cap P_j$$

which implies $f_j(\mathbf{x}_\epsilon^j) \geq (1-\epsilon)f_j(\mathbf{x}^*) = f^*$. Now, $f(\mathbf{x}_\epsilon) \geq f_j(\mathbf{x}_\epsilon^j)$ and thus $f(\mathbf{x}_\epsilon) \geq (1-\epsilon)f^*$. The result then follows. □

3.4.1 Social-welfare maximizing equilibria

When there are many equilibrium configurations in a symmetric game, one may ask which configuration is "best" from a social-welfare point of view. That is, we are interested in finding an optimal solution to the optimization problem:

$$w(\mathbf{x}) = \max\{\sum_{a\in A} x_a u_a(\mathbf{x}) : \mathbf{x} \in N\}; \tag{3.8}$$

that is, find an equilibrium which maximizes the sum of utilities. Let w^* denote its optimal value. The next result describes an FPTAS for this optimization problem.

Theorem 3.4.3. *Consider a symmetric game with either* PWL *or* DPLC *utilities given by the same input as in Theorems 3.3.2 and 3.3.4 respectively and in addition a positive rational number* $\frac{1}{\epsilon}$ *encoded in unary encoding. Then, for a fixed number of actions* m *there exists a polynomial time algorithm to output, in binary encoding, a configuration* $\mathbf{x}_\epsilon \in D$ *that satisfies*

$$w(\mathbf{x}_\epsilon) \geq (1-\epsilon)w^* \text{ where } w(\mathbf{x}) = \sum_{a \in A} x_a u_a(\mathbf{x}) \text{ and } w^* = \max_{\mathbf{x} \in N} w(\mathbf{x})$$

Proof. We establish the result in both the PWL and DPLC settings. We begin with considering PWL utility functions. Partition the feasible region N into subregions where $u_a(\mathbf{x})$ is linear simultaneously for all $a \in A$. This is achieved by considering the problem separately within each cell of the common refinement of each partition $\{P_{aj}\}_{j \in J_a}$ of D. Let $J = \prod_{a \in A} J_a$. This yields the following fine partition of: $D = \bigcup_{\mathbf{j} \in J} P_{\mathbf{j}}$ where $\mathbf{j} = (j_a)_{a \in A} \in J$ and $P_{\mathbf{j}} = \bigcap_{a \in A} P_{aj_a}$. Note that this is also a partition of N. It is then clear that for $\mathbf{x} \in P_{\mathbf{j}}$ each action's utility is $u_a(\mathbf{x}) = f_{aj_a}(\mathbf{x})$ and thus,

$$\begin{aligned} w^* &= \max \left\{ \sum_{a \in A} x_a u_a(\mathbf{x}) : \mathbf{x} \in N \right\} \\ &= \max_{\mathbf{j} \in J} \max \left\{ \sum_{a \in A} x_a f_{aj_a}(\mathbf{x}) : \mathbf{x} \in N \cap P_{\mathbf{j}} \right\}. \end{aligned}$$

We immediately see that this is an optimization of a piecewise-polynomial function over our set N. The result then follows by applying Lemma 3.4.2.

For the DPLC case, by noting the bijection in Proposition 3.3.3 we can view the opti-

mization problem (3.8) as an optimization problem over $\hat{N}$ since:

$$\begin{aligned} w^* &= \max\left\{\sum_{a\in A} x_a u_a(\mathbf{x}) : \mathbf{x} \in N\right\} \\ &= \max\left\{\sum_{a\in A} x_a u_a(\mathbf{x}) : \boldsymbol{\varphi}(\mathbf{x}) = (\mathbf{x},\mathbf{y},\mathbf{z}) \in \hat{N}\right\} \\ &= \max\left\{\sum_{a\in A} x_a (y_{aa} - z_{aa}) : (\mathbf{x},\mathbf{y},\mathbf{z}) \in \hat{N}\right\}. \end{aligned}$$

The latter optimization problem is to maximize the polynomial $\hat{w}(\mathbf{x},\mathbf{y},\mathbf{z}) = \sum_{a\in A} x_a(y_{aa} - z_{aa})$ over the lattice point set $\hat{N}$. Since we assumed that utility are always positive, this is clearly a positive polynomial over the set $\hat{N}$.

Now, apply Lemma 1.4.11 with the following input:

1. Two vectors $(\mathbf{x}^{\mathrm{L}},\mathbf{y}^{\mathrm{L}},\mathbf{z}^{\mathrm{L}}) = \mathbf{0}$ and $(\mathbf{x}^{\mathrm{U}},\mathbf{y}^{\mathrm{U}},\mathbf{z}^{\mathrm{U}})$ where for all $a, a' \in A$, $x_a^{\mathrm{U}} = n$, $y_{aa'}^{\mathrm{U}} = \max_a U_a$ and $z_a^{\mathrm{U}} = \max_a V_a$ (where U_a and V_a are defined as in the proof of Theorem 3.3.5).

2. The data describing the short rational generation function encoding of $\hat{N}$ given by Theorem 3.3.4, where clearly $\hat{N} \in \{(\mathbf{x},\mathbf{y},\mathbf{z}) : (\mathbf{x}^{\mathrm{L}},\mathbf{y}^{\mathrm{L}},\mathbf{z}^{\mathrm{L}}) \leq (\mathbf{x},\mathbf{y},\mathbf{z}) \leq (\mathbf{x}^{\mathrm{U}},\mathbf{y}^{\mathrm{U}},\mathbf{z}^{\mathrm{U}})$.

3. The polynomial objective $\sum_{a\in A} x_a(y_{aa} - z_{aa})$, where the coefficients are all 1 and the exponent vectors are binary.

4. Positive rational number $\frac{1}{\epsilon}$

The output is a point $(\mathbf{x}_\epsilon,\mathbf{y}_\epsilon,\mathbf{z}_\epsilon) \in \hat{N}$ that satisfies $\hat{w}(\mathbf{x}_\epsilon,\mathbf{y}_\epsilon,\mathbf{z}_\epsilon) \geq (1-\epsilon)w^*$. By the bijection we have $\varphi^{-1}(\mathbf{x}_\epsilon,\mathbf{y}_\epsilon,\mathbf{z}_\epsilon) = \mathbf{x}_\epsilon$ and it follows that $w(\mathbf{x}_\epsilon) \geq (1-\epsilon)w^*$, as required. □

One might again ask why we developed a separate argument for the DPLC case, when by Proposition 3.2.6 we could simply convert a DPLC instance into a PWL instance and then use the algorithm in that setting. Note that in showing the result in the two settings,

PWL and DPLC utilities, have two very different approaches. In the PWL case we have to find many ϵ-optimal solutions to subproblems via our partitioning, and in the DPLC case we need to expand the dimension of sets under consideration using an extended formulation. As our methods are sensitive to dimension the latter seems unfavorable, but a single FPTAS is required in this setting and so computational advantage of one method overall another is somewhat ambiguous. Further considerations of this nature are a potential avenue for future investigations.

3.4.2 Mean and variance of social welfare

A related question to social welfare maximization is that of calculating the mean and variance of social welfare over all equilibrium configurations. Much attention has been given the case of minimum and maximum welfare associated with equilibria, in the context of price and anarchy and stability, respectively (see for example [3, 71]). The *pure price of anarchy* is the ratio of the *worst* social welfare that arises from a PSNE to the maximum social welfare where the agents act together to maximize their total payoff. The *pure price of stability* is the ratio of the *best* social welfare that arises from a PSNE to the maximum social welfare. We are in search of an average welfare over all equilibria, which we could use, for instance, to compute another notion: the average price of anarchy. Also, being able to compute the variance of utilities may be useful. A mechanism designer, for instance, may prefer games with smaller variance in welfare in order to ensure some regularity in the welfare conditions of a game.

At first glance the ability to compute the mean and variance of social welfare would appear to be computationally expensive since in the worst case there may be an exponential number of equilibrium configurations. Nonetheless, we show that our polynomial-sized short rational generating function encoding of the set of equilibrium configurations can be used to answer this question efficiently. We isolate discussion to the PWL case.

Given a symmetric game we let μ_w and σ_w^2 denote the mean and variance of social welfare over all its equilibrium configurations N which can be expressed as:

$$\mu_w = \frac{\sum_{\mathbf{x}\in N} w(\mathbf{x})}{|N|} \tag{3.9}$$

and

$$\sigma_w^2 = \frac{\sum_{\mathbf{x}\in N}(w(\mathbf{x}) - \mu_w)^2}{|N|}.$$

We claim that μ_w and σ_w^2 can be computed in polynomial time when the number is actions is fixed.

Theorem 3.4.4. *Consider a symmetric game with either* PWL *or* DPLC *utilities given by the same input as in Theorems 3.3.2 and 3.3.4 respectively. Then, when the number of actions is fixed, there exists a polynomial time algorithms to find the mean social welfare μ_w and variance of social welfare σ_w^2 over all equilibrium configurations.*

Proof. We show how to compute μ_w in polynomial time by establishing algorithms to compute the numerator and denominator of the right-hand side of (3.9). The denominator $|N|$ can be computed using the Counting Lemma (Lemma 1.4.8) applied to the Barvinok encoding of N. We calculate the numerator by partitioning D as in the proof of Theorem 3.4.3 and breaking up the sum into smaller whole sums over the cells in the partition; that is, by computing

$$\sum_{\mathbf{x}\in N} \hat{w}(\mathbf{x}) = \sum_{\mathbf{j}\in J} w_{\mathbf{j}} \tag{3.10}$$

where

$$w_{\mathbf{j}} = \sum_{\mathbf{x}\in N\cap P_{\mathbf{j}}} \sum_{a\in A} x_a f_{aj_a}(\mathbf{x}) \tag{3.11}$$

is the partial sum of utilities in cell $N \cap P_{\mathbf{j}}$ using the same notation as in the proof of Theorem 3.4.3.

To compute this sum we compute each $w_{\mathbf{j}}$ separately. Note that $\sum_{a \in A} x_a f_{aj_a}(\mathbf{x})$ is a polynomial function over the set $N \cap P_{\mathbf{j}}$ (which as have shown in the proof of Theorem 3.3.2 has a short rational generating functioning encoding) and thus by Lemma 1.4.12 we can evaluate the sum. This in turn yields an overall polynomial time algorithm to compute μ_w.

A similar argument holds for finding σ_w^2 by noting that the function $f(\mathbf{x}) = (w(\mathbf{x}) - \mu_w)^2$ is a polynomial function over each cell $N \cap P_{\mathbf{j}}$ of the partition of N and then appealing to Lemma 1.4.12. □

Note that we can evaluate the mean and standard deviation of social welfare without recourse to enumeration of any equilibrium configurations, which again underscores the power of encoding our sets of interest as rational generating functions.

3.4.3 Distribution of payoffs in equilibrium

It should be noted that Theorem 3.4.4 can be generalized to hold for finding the mean and variance of evaluating an arbitrary polynomial function $f(\mathbf{x})$ on the elements of N.

One such function of interest is simply the payoff $u_a(\mathbf{x})$ for players playing action a in configuration $\mathbf{x}$. In this subsection we explore the distribution of payoffs $u_a(\mathbf{x})$ for playing action a amongst all equilibrium configurations where actions a is played, i.e. for $\mathbf{x} \in N$ with $x_a \geq 1$. We let N_a denote this set of equilibria. We immediately get the following result.

Proposition 3.4.5. *Consider a symmetric game with either* PWL *or* DPLC *utilities given by the same input as in Theorems 3.3.2 and 3.3.4 respectively. Then, when the number of actions is fixed, for each specified action a there exists a polynomial time algorithm to compute the mean μ_a and variance σ_a^2 of payoffs for playing action a, taken over all equilibrium configurations $\mathbf{x} \in N_a$.*

The proof is analogous to the proof of Theorem 3.4.4, with the only significant difference being that the we exchange N_a for N. Details are omitted.

However, since $u_a(\mathbf{x})$ corresponds to an *affine* functions over the cells $N_a \cap P_{aj}$ there is still more that we can say about how its values are distributed. As will be seen in our next result, we can construct a frequency histogram to describe the distribution of payoffs $u_a(\mathbf{x})$ over N_a.

The key step is to *count* equilibria in N_a where the payoffs to players playing action a falls in some range $u_i \leq u_a(\mathbf{x}) < u_{i+1}$ where $u_i < u_{i+1}$ are integers. The histogram $H_a^{u_0,u_1,\ldots,u_T}$ will consist of counts of equilibria lying in consecutive intervals $[u_i, u_i + 1)$ for $i = 0, \ldots T-2$ and $[u_{T-1}, u_T]$ with

$$u_{\min} = u_0 < u_1 < \cdots < u_{T-1} < u_T = u_{\max}$$

where

$$u_{\min} = \min\{u_a(\mathbf{x}) : \mathbf{x} \in N_a\},$$

$$u_{\max} = \max\{u_a(\mathbf{x}) : \mathbf{x} \in N_a\}$$

Let $c_{i,i+1}$ denote the number of equilibria in N_a where $u_a(\mathbf{x})$ lies in the interval $[u_i, u_{i+1})$ for $i = 0, \ldots T-2$ and the interval $[u_{T-1}, u_T]$ for $i = T-1$.

The next result shows such a histogram can be produced in polynomial time when the number of intervals T is polynomial in the input size.

Theorem 3.4.6. *Consider a symmetric game with either* PWL *or* DPLC *utilities given by the same input as in Theorems 3.3.2 and 3.3.4 respectively. Fix the number of actions m. There exists a polynomial time algorithm to construct histogram $H_a^{u_0,u_1,\ldots,u_T}$ provided the number of consecutive intervals T for payoffs is of size polynomial in the binary encoding of the input.*

Proof. As in the previous theorem our approach is to compute counts for the cells $N_a \cap P_{aj}$

then to sum the results over all $j \in J_a$. Thus, for $i \neq 0, T-1$ we have

$$c_{i,i+1} = \sum_{j \in J_a} |\{\mathbf{x} \in N \cap P_{aj} : u_i \leq f_{aj}(\mathbf{x}) \leq u_{i+1}\}|.$$

To compute the cardinalities of the sets in the sum we apply the Counting Lemma (Lemma 1.4.8) to the rational generating function of $\{\mathbf{x} \in N \cap P_{aj} : u_i \leq f_{aj}(\mathbf{x}) \leq u_{i+1}\}$ which itself can be found in polynomial time using the Boolean Operations Lemma (Lemma 1.4.4).

It remains to show how to compute $c_{0,1}$ and $c_{T-1,T}$, and this is achieved as above once we know $u_{\min}$ and $u_{\max}$. We only give details for finding $u_{\max}$. Again using our partition of N_a it suffices to compute $\max\{f_{aj}(\mathbf{x}) : \mathbf{x} \in N \cap P_{aj}\}$ for each $j \in J_a$. Given our short rational generating function encoding of $N \cap P_{aj}$ this can be found in polynomial time by an appeal to the Linear Optimization Lemma (Lemma 1.4.10) with the appropriate input. □

3.5 M-symmetric games

A game is M-symmetric, when the set of players can be partitioned into M classes, labeled $\{1, \ldots, M\}$, such that within each class, players are interchangeable. This means that players in each class have the same number of actions and the same utility functions. Formally, each class i has n_i players and A_i actions with $A = \bigcup_{i=1}^{M} A_i$.

Define an M-configuration to be a vector $\mathbf{x} \in Z^A$ such that for each $i \in \{1, \ldots, M\}$, for each $a \in A_i$, x_{ia} is the number of players in class i that choose action a. The set of M-configurations is $D = \prod_{i=1}^{M} D_i$ where

$$D_i = \{\mathbf{x}_i \in Z^{|A_i|} : \mathbf{x}_i \geq 0, \sum_{a \in A_i} x_{ia} = n_i\}.$$

For a player from class i, her utility of playing action $a \in A_i$ is a function of the configuration $\mathbf{x}$, denoted $u_{ia}(\mathbf{x})$.

We are interested in M-symmetric games with piecewise linear utility functions. The input is as follows: for each class i there is a polyhedral subdivision $D_i = \bigcup_{j \in J_{ia}} P_{iaj}$ where J_{ia} is a finite index set, over affine functions f_{iaj} for $j \in J_{ia}$ defining utility functions

$$u_{ia}(\mathbf{x}) = f_{iaj}(\mathbf{x}) \text{ for } \mathbf{x} \in P_{iaj}.$$

An M-configuration $\mathbf{x}$ is a pure Nash equilibrium M-configuration if and only if for all $i \in \{1, \ldots, M\}$ and for all $a, a' \in A_j$, $x_{ia} = 0$ or $u_{ia}(\mathbf{x}) \geq u_{ia'}(\mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a)$. Using similar arguments as before, we can encode the set of pure equilibria as a short rational generating function. This procedure works in polynomial time in the input size (that is, the binary encoding of n and the description of the objective functions) as long as m and M are bounded constants.

3.6 Parameterized symmetric games

In this section we extend our approach to model families of symmetric games parameterized by integer parameters, and answer interesting questions about a family of games without having to solve each individual game in the family. Many problems in mechanism design have such a flavor; e.g. designing a game (i.e., setting the parameters) such that the resulting equilibria satisfy certain properties or optimize a certain objective.

We allow a fixed number of integer parameters that additively influence each piece of the utility functions, and furthermore allow the *number* of players to be considered as a parameter. The set of actions remains fixed in each member of the parametric family.

Our overall approach is to use rational generating functions as before, except that we augment the configuration vector with the vector of integer parameters. The resulting set we encode as a generating function is the graph of the parameters–equilibria correspondence.

3.6.1 Parametric families of symmetric games

Since we are considering parametric families of games in which the number of players is a changing parameter, we define our piecewise linear utilities over configurations of different numbers of players. That is, we define the utilities over the set of *feasible configurations*

$$F = \{\mathbf{x} \in \mathbb{Z}^m : 1 \leq \sum_{a \in A} x_a \leq B, x_a \geq 0, \ \forall a \in A\},$$

where B is a bound on the total number of players. In other words, the number of players in the game can be between 1 and B.

The utilities are specified by a polyhedral subdivision $\{P_{aj}\}_{j \in J_a}$ of F for each action a, and affine functions f_{aj} defined over the cells P_{aj} as in Section 3.2 except that now the division is over F. We also introduce a fixed number of parameters that control the additive constants in the affine functions f_{aj}. We constrain these parameters to be the lattice points inside a polytope of fixed dimension $R \in \mathbb{R}^d$. Formally, let $\mathbf{p}$ be a d-dimensional integer vector inside of $Q = R \cap \mathbb{Z}^d$ and for each $a \in A$ and $j \in J_a$, define

$$f_{aj}(\mathbf{x}, \mathbf{p}) = \boldsymbol{\alpha}_{aj} \cdot \mathbf{x} + \boldsymbol{\beta}_{aj} \cdot \mathbf{p}, \tag{3.12}$$

where now

$$u_a(\mathbf{x}, \mathbf{p}) = f_{aj}(\mathbf{x}, \mathbf{p}) \text{ for } \mathbf{p} \in Q \text{ and } \mathbf{x} \in P_{aj}. \tag{3.13}$$

In other words, the parameter allows for an additive change in each piece of the piecewise linear utility function u_a.

We are interested in encoding the set of *parameterized* PSNE defined as

$$N = \{(\mathbf{x}, n, \mathbf{p}) : \mathbf{p} \in Q, n \in \{1, \ldots, B\}, \mathbf{x} \in N(n, \mathbf{p})\}, \tag{3.14}$$

where $N(n, \mathbf{p})$ is the set of PSNE in the symmetric game with n players, feasible configurations $D_n = \{\mathbf{x} \in F : \sum_{a \in A} x_a = n\}$ and utilities defined as in (3.13) for the given parameter $\mathbf{p}$.

The approach we use to encode N is similar to that of Section 3.3 in that we describe N as a Boolean combination of sets of lattice points contained in polyhedra that in part derive from a partitioning of N into cells of the form $N \cap P_{\mathbf{j}}$ using the same notation as in the proof of Theorem 3.4.3. The expression of N as a Boolean combination of sets is

$$N = H \setminus \bigcup_{a,a'} \biguplus_{j} \biguplus_{j'} Dev(a, a', j, j'), \tag{3.15}$$

where

$$H = \{(\mathbf{x}, n, \mathbf{p}) : \mathbf{x} \in D_n, n \in \{1, \ldots, B\},\ \mathbf{p} \in Q\}$$

is the overall feasible set, and the first union is over all $a, a' \in A$, the second union is disjoint over $j \in J_a$, the third union is disjoint over $j' \in J_{a'}$ and $Dev(a, a', j, j')$ is the set

$$Dev(a, a', j, j') = \left\{ (\mathbf{x}, n, \mathbf{p}) : \begin{array}{l} \mathbf{p} \in Q,\ 1 \leq n \leq B, n \in \mathbb{Z} \\ \mathbf{x} \in D_n \cap P_{aj} \\ \mathbf{x}' = \mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a \in P_{a'j'} \\ f_{aj}(\mathbf{x}) \leq f_{a'j'}(\mathbf{x}') - 1\} \end{array} \right\}. \tag{3.16}$$

It is clear that H and each $Dev(a, a', j, j')$ is a polytopal lattice point set in $(\mathbf{x}, n, \mathbf{p})$ and thus a Barvinok encoding of N obtains by applying similar reasoning to that found in Theorem 3.3.1.

We have thus established the following.

Theorem 3.6.1. *Consider a parametric family of symmetric games with m actions and n players with utility functions parameterized by $\mathbf{p} \in Q$ as found in* (3.13) *given by the*

following input in binary encoding:

(I_1) *an integer bound B on the number of players;*

(I_2) *an inequality description of a rational polytope R contained in $\mathbb{R}^d$;*

(I_3) *for each $a \in A$, a nonnegative integer $|J_a|$ and each $j \in J_a = \{1, \ldots, |J_a|\}$ an inequality description of the polytope P_{aj} and integer vectors $\boldsymbol{\alpha}_{aj} \in \mathbb{Z}^m$, and integers $\boldsymbol{\beta}_{aj}$ defining the affine functions $f_{aj}(\mathbf{x}) = \boldsymbol{\alpha}_{aj} \cdot \mathbf{x} + \boldsymbol{\beta}_{aj} \cdot \mathbf{p}$.*

Then, the set N of parameterized PSNE *defined in* (3.14) *has a Barvinok encoding, which can be computed in polynomial time when the number of actions m and the dimension of the parameter space d is fixed.*

3.6.2 Optimization over parameters

Once we have a Barvinok encoding of the set N of parameterized PSNE, we can use it to answer questions about game-theoretic properties of the parameterized family. To give a basic illustration, suppose we are interested in finding a value for the parameters $(n, \mathbf{p})$ such that a PSNE exists. This is resolved by enumerating from the set N and terminating after the first element $(\mathbf{x}, n, \mathbf{p})$ is output. A more sophisticated question is to optimize linear and nonnegative polynomial objective functions of the parameters, subject to the constraint that a PSNE exists. Formally, for minimizing objective function f we solve: $\min\{f(n, \mathbf{p}) : (\mathbf{x}, n, \mathbf{p}) \in N\}$. We can solve problems of this type by appeal to Lemma 3.4.1 (for piecewise linear objectives) or Lemma 3.4.2 (for nonnegative piecewise polynomial objectives).

A more interesting case is when we want to find optimal parameters under some objective function that depends on the PSNE configuration as well as on the parameters themselves. Since each game in the family can have multiple PSNEs, the problem is not well defined until we specify how PSNEs are selected. Depending on the application domain, we might consider

selecting the best-case PSNE for each game, or selecting the worst-case PSNE. By "best-case" PSNE we mean that for a given parameter instantiation $(n, \mathbf{p})$ the mechanism designer considers the most favorable PSNE in $N(n, \mathbf{p})$ when attempting to optimize its objective. For example, in mechanism design problems, best-case analysis (e.g. recent work on the price of stability [3]) is useful for getting bounds on what we could achieve under equilibrium behavior. By "worst-case" PSNE we mean that for a given parameter instantiation $(n, \mathbf{p})$ the mechanism designer the least favorable PSNE in $N(n, \mathbf{p})$. Worst-case analysis (e.g. recent work on the price of anarchy [54, 71]) is useful for designing mechanisms with guaranteed performance.

Depending on whether we want to select best-case PSNE or worst-case PSNE, different types of optimization problems arise. For example, if we want to minimize the function $f(\mathbf{x}, n, \mathbf{p})$, selecting the best-case PSNE yields the problem

$$\min\{f(\mathbf{x}, n, \mathbf{p}) : (\mathbf{x}, n, \mathbf{p}) \in N\}. \tag{3.17}$$

When f is piecewise linear or piecewise polynomial, this can be solved in polynomial time or approximated in polynomial time[5], respectively, using Lemma 3.4.1 and Lemma 3.4.2.

We now turn to the more delicate situation of selecting the worst-case PSNE. This yields the problem

$$\min_{n,\mathbf{p}} \quad \max_{\mathbf{x}:(\mathbf{x},n,\mathbf{p})\in N} \quad f(\mathbf{x}, n, \mathbf{p}). \tag{3.18}$$

The decision version of the problem is the following: decide whether there exists a parameter instantiation $(n, \mathbf{p})$ such that $N(n, \mathbf{p}) \neq \emptyset$ and for all $\mathbf{x} \in N(n, \mathbf{p})$, $f(\mathbf{x}, n, \mathbf{p}) \geq v$. This problem has a similar form to the parameterized integer programming problem studied by [37]. However, applied to our domain, the approach in [37] requires that the feasible set of $\mathbf{x}$

[5]Throughout Section 3.6 the phrase "polynomial time" is always understood to be under the caveat that the number of actions m is fixed.

be the set of integer points in a convex polytope. This is not the case for our problem since the set $N(n, \mathbf{p})$ depends on the parameters and is not described by a convex polytope.

We propose a branch-and-bound approach for solving the min-max optimization (3.18). Branch and bound is a general optimization method for nonconvex optimization problems, applicable to discrete domains as well as continuous domains [57, 62]. At a high level, the algorithm partitions the space of candidate solutions into regions and iteratively refines the partitioning by bisecting a region along one of its dimensions. The algorithm can prune off regions using bounds on the objective value for each of the regions. There is a wide choice on the order in which the regions are explored. For discrete domains, a simple and memory-efficient approach is to explore the regions in a depth-first order. The steps of a simple branch-and-bound algorithm for (3.18) is outlined as follows:

1. Set of regions $\mathcal{L}$ is initialized with one region $\{(n, \mathbf{p}) : 0 \leq n \leq B, \mathbf{p} \in R\} \cap \mathbb{Z}^{d+1}$.
2. Global upper bound UB is initialized to ∞.
3. Repeat until $\mathcal{L}$ is empty:
   (a) Take the next region L from $\mathcal{L}$ in depth-first order.
   (b) If L contains only one integer point $(\hat{n}, \hat{\mathbf{p}})$, solve $\hat{f} = \max_{\mathbf{x}:(\mathbf{x},\hat{n},\hat{\mathbf{p}})\in N} f(\mathbf{x}, \hat{n}, \hat{\mathbf{p}})$. Update UB if $\hat{f} > UB$.
   (c) Otherwise (i.e., L contains more than one integer point), compute the lower bound LB for L as discussed above. If $LB \geq UB$ discard L. Otherwise bisect L along its longest dimension, and replace L in $\mathcal{L}$ by the resulting two regions.
4. Return UB as the optimal value.

For our problem, each "region" corresponds to the set of integer points in a polytope. The space of candidate solutions $(n, \mathbf{p})$ is $\{(n, \mathbf{p}) : 0 \leq n \leq B, \mathbf{p} \in R\} \cap \mathbb{Z}^{d+1}$. This can be

bisected into two regions; for example, by adding constraints $n \leq \lfloor B/2 \rfloor$ and $n \geq \lfloor B/2 \rfloor + 1$ respectively.

For the branch-and-bound approach to be effective it requires that each iteration of Step 3 can be performed efficiently; that is, the bounds on the objective value can be efficiently computed for each region. This is indeed the case in our setting:

Proposition 3.6.2. *Each iteration of Step 3 executes in polynomial time when f is a piecewise linear function as defined in* (3.12).

Proof. For each region L, we compute a lower bound on the value of any candidate solution $(n, \mathbf{p})$ from the region by solving

$$\min\{f(\mathbf{x}, n, \mathbf{p}) : (\mathbf{x}, n, \mathbf{p}) \in N, (n, \mathbf{p}) \in L\}. \tag{3.19}$$

Since L is a polytope at each iteration, we derive a Barvinok encoding of the feasible set of (3.19) and apply Lemma 3.4.1.

We compute an upper bound by picking some $(\hat{n}, \hat{\mathbf{p}})$ in L and solving the optimization $\max_{\mathbf{x}:(\mathbf{x},\hat{n},\hat{\mathbf{p}})\in N} f(\mathbf{x}, \hat{n}, \hat{\mathbf{p}})$. Similar to above, these bounds can be computed in polynomial time if f is piecewise linear. □

Thus, for piecewise-linear f, this algorithm is guaranteed to find the optimum in finite time, although the worse-case running time is exponential. Nevertheless, when the algorithm can prune off large regions, it can be much faster than the brute-force approach of trying every parameter instantiation.

3.6.3 Fitting a "closest game" to an observed equilibrium

We now consider a problem from econometrics: determining parameters of a utility model that would give rise to observed equilibrium behavior. Suppose we are analyzing a system

modeled as an n-player symmetric game where certain aspects of the game's utility functions are unknown, and furthermore the set of candidate utility functions belongs to the set of parametric (in $\mathbf{p}$) piecewise-linear utilities of the form of (3.13).

We are given the players' observed behavior in the game as some configuration $\tilde{\mathbf{x}}$. Assuming the players are behaving rationally, we would like to know the game's true utility functions, i.e., to estimate the parameter $\mathbf{p}$. Note that we are not parameterizing the number of players n, since given any observed configuration $\tilde{\mathbf{x}}$ we automatically determine the number of players n as $\sum_{a \in A} \tilde{x}_a$. Thus, for simplicity we use the notation $N(\mathbf{p})$ instead of $N(n, \mathbf{p})$ to denote the set of PSNEs with parameter $\mathbf{p}$.

Assuming the players play a PSNE if one exists, we would like to find a $\mathbf{p} \in Q$ such that $\tilde{x}$ is a PSNE of the game with parameter $\mathbf{p}$. Formally, the set of such $\mathbf{p}$ is

$$N_{\tilde{\mathbf{x}}} = \{\mathbf{p} : \mathbf{p} \in Q, (\tilde{\mathbf{x}}, \mathbf{p}) \in N\}.$$

A Barvinok encoding of $N_{\tilde{\mathbf{x}}}$ can be computed efficiently using a similar construction as in Section 3.6.1 with the additional constraint $\mathbf{x} = \tilde{\mathbf{x}}$.

However, there might not be such a $\mathbf{p}$, i.e., the set $N_{\tilde{\mathbf{x}}}$ might be empty. In this case we would like to find $\mathbf{p}$ such that $\tilde{\mathbf{x}}$ is close to being an equilibrium. One measure of approximate equilibrium is an ϵ-equilibrium. Note that in the symmetric games we are considering, the utility functions are integer-valued at integer points in their domain, and thus we may restrict ϵ to be integer for our purposes. Note that we can impose an upper bound on the ϵ's of interest, a simple bound being the largest utility value possible amongst all actions over the set D. That is,

$$u_{max} = \max_{a \in A} \max\{u_a(\mathbf{x}) : \mathbf{x} \in D\} \tag{3.20}$$

which can be solved in polynomial time (integer programming using Lenstra's algorithm [58]) when the number of actions m is fixed.

Define $\tilde{N}$ to be the set of $(\mathbf{x}, \mathbf{p}, \epsilon)$ such that $\mathbf{x}$ is an ϵ-PSNE of the game with parameter $\mathbf{p}$. We can compute a Barvinok encoding of $\tilde{N}$ by adapting the proof of Theorem 3.6.1 as follows: make ϵ an additional integer variable and redefine the deviation set $Dev(a, a', j, j')$ in (3.16) to be:

$$Dev(a, a', j, j') = \left\{ (\mathbf{x}, \mathbf{p}, \epsilon) : \begin{array}{l} \mathbf{p} \in Q, 0 \leq \epsilon \leq u_{max}, \epsilon \in \mathbb{Z} \\ \mathbf{x} \in D_n \cap P_{aj} \\ \mathbf{x}' = \mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a \in P_{a'j'} \\ f_{aj}(\mathbf{x}) \leq f_{a'j'}(\mathbf{x}') - \epsilon - 1\} \end{array} \right\}.$$

Then finding the best parameter $\mathbf{p}$ amounts to solving the optimization

$$\begin{array}{ll} \min & \epsilon \\ \text{s.t.} & (\mathbf{x}, \mathbf{p}, \epsilon) \in \tilde{N},\ \mathbf{x} = \tilde{\mathbf{x}}. \end{array}$$

This can be solved in polynomial time using the Linear Optimization Lemma (Lemma 1.4.10) for linear objective ϵ over the set

$$\tilde{N} \cap \{(\mathbf{x}, \mathbf{p}, \epsilon) : \mathbf{p} \in Q, \epsilon \in \mathbb{Z}, \mathbf{x} = \tilde{\mathbf{x}}, 0 \leq \epsilon \leq u_{max}, \}$$

which admits a Barvinok encoding by appeal to the Boolean Operations Lemma (Lemma 1.4.4).

Instead of using ϵ-equilibria, we could try to minimize the distance between $\tilde{\mathbf{x}}$ and the set $N(\mathbf{p})$ of PSNE given $\mathbf{p}$, which is defined as the infimum of distances between $\tilde{\mathbf{x}}$ and points in the set $N(\mathbf{p})$. This yields the optimization problem

$$\begin{array}{ll} \min & d(\mathbf{x}, \tilde{\mathbf{x}}) \qquad (3.21) \\ \text{s.t.} & (\mathbf{x}, \mathbf{p}) \in N, \qquad (3.22) \end{array}$$

where d is the distance metric. Section 4 of De Loera et al. [23] describes an analogous scenario. Their results, with minor adjustments, apply to our setting. De Loera et al. consider the problem of finding the closest Pareto optimum of a multicriteria integer program to some specified point $\hat{\mathbf{v}}$. The set of Pareto optima V can be expressed in Barvinok encoding when certain parameters are fixed, and when considering certain metrics d they find closest points in V to $\hat{\mathbf{v}}$ efficiently. For ℓ_1- and ℓ_∞-norm induced metrics, problem (3.21) can be solved in polynomial time [23, Theorem 10]. For ℓ_p-norm induced metrics ($p \geq 1$) there exists an FPTAS [23, Theorem 13 and Remark 15].

Another scenario is when the set $N_{\tilde{\mathbf{x}}}$ has multiple points and we would like to find the parameter $\mathbf{p}$ with the "best fit". The distance between $\tilde{x}$ and $N(\mathbf{p})$ is 0 for $\mathbf{p} \in N_{\tilde{\mathbf{x}}}$, and is thus not useful here. By "best fit" we mean the $\mathbf{p} \in N_{\tilde{\mathbf{x}}}$ such that $\tilde{\mathbf{x}}$ is closest to all points in $N(\mathbf{p})$. In other words, we choose the $\mathbf{p}$ that minimizes the largest distance from $\tilde{\mathbf{x}}$ to any equilibrium in $\mathbf{x} \in N(\mathbf{p})$. Formally, we solve

$$\min_{\mathbf{p} \in N_{\tilde{\mathbf{x}}}} \max_{\mathbf{x}\,\text{s.t.}\,(\mathbf{x},\mathbf{p}) \in N} d(\mathbf{x}, \tilde{\mathbf{x}}). \tag{3.23}$$

This is an instance of the min-max optimization problem (3.18). When d is the ℓ_1 or ℓ_∞ distance, $d(\mathbf{x}, \tilde{\mathbf{x}})$ is piecewise linear in $\mathbf{x}$, in which case the problem (3.23) can be solved using the branch-and-bound algorithm as discussed in Section 3.6.2. To avoid having to partition the set $N_{\tilde{\mathbf{x}}}$, we can instead let $\mathbf{p}$ range from Q, and modify Step 3c of the algorithm so that whenever we have a region $L \subset Q$ such that $L \cap N_{\tilde{\mathbf{x}}} = \emptyset$, we set the lower bound for L to be ∞.

3.6.4 Finding parameters with good equilibrium payoffs

A common goal in mechanism design is to set the parameters of the game so as to achieve some desired properties in the resulting equilibria. Here we consider a mechanism designer

choosing the parameters n and $\mathbf{p}$ such that the resulting symmetric game has equilibria with good payoffs. There are several ways to measure the "goodness" of an equilibrium. Suppose the objective we are interested in is the payoff of the worst-off player given an equilibrium $\mathbf{x}$, i.e., $\min_{a:x_a>0} u_a(\mathbf{x}, \mathbf{p})$. As discussed in Section 3.6.2, we need to specify how PSNE are selected for each game. If we pick the best-case PSNE, this yields the optimization problem

$$\max_{(\mathbf{x},n,p)\in N} \min_{a:x_a>0} u_a(\mathbf{x}, \mathbf{p}). \tag{3.24}$$

The next result shows that this can be solved in polynomial time.

Theorem 3.6.3. *Consider a parametric family of symmetric games with m actions and n players with identical input as in Theorem 3.6.1. Problem* (3.24) *of finding the best-case* PSNE *which maximizes the worst-case payoff to a player, can be solved in polynomial time when m is fixed.*

Proof. The problem can be rewritten as

$$\max \quad w \tag{3.25}$$

$$\text{s.t.} \quad (\mathbf{x}, n, \mathbf{p}, w) \in S \tag{3.26}$$

where S is the set

$$\left\{ (\mathbf{x}, n, \mathbf{p}, w) : \begin{array}{l} (\mathbf{x}, n, \mathbf{p}) \in N,\ u_{\min} \le w \le u_{\max} \\ \forall a \in A : w \le u_a(\mathbf{x}, \mathbf{p}) \text{ OR } x_a = 0 \end{array} \right\}.$$

$u_{\min}$ and $u_{\max}$ are the minimum and maximum utilities possible in the game. They can be computed in polynomial time in fixed dimension (see (3.20) above). We use a similar technique as in Section 3.6.1 to construct a Barvinok encoding of S in polynomial time. It

is straightforward to verify that we can write:

$$S = G \setminus \left(\bigcup_{a,a'} \biguplus_{j} \biguplus_{j'} Dev(a, a', j, j') \right) \setminus \left(\bigcup_{a} U_a \right)$$

where G is the overall feasible set

$$G = \left\{ (\mathbf{x}, n, \mathbf{p}, w) : \begin{array}{l} \mathbf{x} \in D_n,\ 0 \leq n \leq B, n \in \mathbb{Z} \\ \mathbf{p} \in Q,\ u_{\min} \leq w \leq u_{\max},\ w \in \mathbb{Z} \end{array} \right\},$$

$Dev(a, a', j, j')$ is the set

$$Dev(a, a', j, j') = \left\{ (\mathbf{x}, n, \mathbf{p}, w) : \begin{array}{l} \mathbf{p} \in Q,\ 0 \leq n \leq B, n \in \mathbb{Z} \\ u_{\min} \leq w \leq u_{\max} \\ \mathbf{x} \in D_n \cap P_{aj} \\ \mathbf{x}' = \mathbf{x} + \mathbf{e}_{a'} - \mathbf{e}_a \in P_{a'j'} \\ f_{aj}(\mathbf{x}) \leq f_{a'j'}(\mathbf{x}') - 1 \end{array} \right\}$$

and $U_a = \{(\mathbf{x}, n, \mathbf{p}, w) : \sum_{a' \in A} x'_a = n, x_a \geq 0, \mathbf{p} \in Q,\ 0 \leq n \leq B,\ w \geq u_a(\mathbf{x}, \mathbf{p}) + 1\}$. The sets U_a ensure that the points that remain in S have $w \leq u_a(\mathbf{x}, \mathbf{p})$.

Based on similar reasoning as above then, we can derive the Barvinok encoding of S in polynomial time. We solve (3.25) by applying the Linear Optimization Lemma (Lemma 1.4.10).

□

If we instead pick the worst-case PSNE, we have the optimization problem

$$\max_{n,p} \quad \min_{\mathbf{x}:(\mathbf{x},n,\mathbf{p}) \in N} \min_{a:x_a > 0} u_a(\mathbf{x}, \mathbf{p}). \tag{3.27}$$

This is an instance of problem (3.18), and we can apply the branch-and-bound algorithm as discussed in Section 3.6.2. It remains to show that Step 3 of the algorithm can be executed

in polynomial time. This is the case if we can efficiently solve the subproblems:

$$\max_{(\mathbf{x},n,\mathbf{p})\in N:(\mathbf{n},\mathbf{p})\in L} \min_{a:x_a>0} u_a(\mathbf{x},\mathbf{p})$$

for each subregion L of $\{(n,\mathbf{p}) : 0 \le n \le B, n \in \mathbb{Z}, \mathbf{p} \in Q\}$, and

$$\min_{\mathbf{x}:(\mathbf{x},\tilde{n},\tilde{\mathbf{p}})\in N} \min_{a:x_a>0} u_a(\mathbf{x},\tilde{\mathbf{p}})$$

for any given $\tilde{n}$ and $\tilde{\mathbf{p}}$. The former has a similar form as the best-case problem (3.24) and can be solved using the same approach as above in polynomial time. Details are omitted.

To solve the latter problem, we break the feasible set into regions: for each subset T of A, we have one region $D(T) = \{x \in D_{\tilde{n}} : x_a > 0 \text{ for all } a \in T, x_a = 0 \text{ for all } a \notin T\}$. There is a constant number of these regions and we solve the problem separately on each region. Formally, we rewrite the problem as

$$\min_{T\subseteq A} \quad \min_{\mathbf{x}\in D(T):(\mathbf{x},\tilde{n},\tilde{\mathbf{p}})\in N} \quad \min_{a\in T} u_a(\mathbf{x},\tilde{\mathbf{p}}).$$

We can now exchange the latter two min operators to get

$$\min_{T\subseteq A} \quad \min_{a\in T} \quad \min_{\mathbf{x}\in D(T):(\mathbf{x},\tilde{n},\tilde{\mathbf{p}})\in N} u_a(\mathbf{x},\tilde{\mathbf{p}}).$$

The innermost minimization can be solved in polynomial time since $u_a(\mathbf{x},\tilde{\mathbf{p}})$ is piecewise-linear. Since the other minimizations are over sets of fixed size, this can be solved in polynomial time.

3.7 Comparison with integer programming games

The method of this chapter shares some important similarities with that of the previous chapter. In particular both build on the theory of rational generating functions, largely from the 2003 paper by Barvinok and Woods [8]. There are, however, some important differences. In the symmetric case, because there are a finite number of actions, it is much easier to capture deviations in this setting. Indeed, as we saw, unilateral deviations will only affect a configuration vector by 1 in two entries and thus there are few deviations to consider. By contrast, in integer programming games, a unilateral deviation can go from the current action profile to a large, in fact exponential, number of different alternate profiles. This is the major reason behind the fact that in this chapter we can avoid the use of the Projection Lemma which seems quite necessary in the integer programming games case.

The fact that we can avoid the Projection Lemma altogether in the symmetric case makes the algorithms of this chapter more likely to be practical than in the previous chapter. Recall in Remark 1.4.13 it was mentioned that the current implementations of the Projection Lemma only run on primitive cases (that is, problems with a small number of variables) and there seems more promise if we only need to do boolean operations, as is the case here.

3.8 Conclusions and directions for further research

In this chapter we explored a promising form of compactly represented symmetric games – those with piecewise linear objectives. Given enough pieces such a setting is completely general, but the methods explored here allow us to PSNE with the complexity scaling in the number of pieces and not the number of configurations which define the game. This is a major advantage over previous methods. We believe no existing compact representations of symmetric games combine both the generality and powerful approach to computation offered here.

Also of note is the novel approach for studying pure strategy Nash equilibria – rational generating functions. The simplicity by which they were used to compute important information on the structure of these games, again reinforces the power of generating functions as an analytical and computational tool for game theory. This confirms the hope of the previous chapter that these methods can be used to enlighten further investigations in algorithmic game theory, and only adds weight to the case for future use of the method.

This work also puts more impetus on searching for improved implementations of Barvinok's Theorem (Lemma 1.4.3) and the Boolean Operations Lemma (Lemma 1.4.4), an important area for future research. The results on parametric games also hold promise as a methodology for further investigations on the nature and structure of symmetric games. Future studies might do well to explore practical computations to glean insights in this direction.

Chapter 4

A parametric integer programming for bilevel mixed integer programs

4.1 Introduction

The main focus of this chapter is the following bilevel mixed integer linear program (BMILP) introduced in Chapter 1, where the leader's variables $\mathbf{z} \in \mathbb{R}^d$ are continuous and the follower's variables $\mathbf{x} \in \mathbb{Z}^n$ are integer:

$$\inf_{\mathbf{x},\mathbf{z}} \quad \mathbf{c} \cdot \mathbf{x} + \mathbf{e} \cdot \mathbf{z} \tag{4.1}$$

$$\text{s.t.} \quad C\mathbf{x} + D\mathbf{z} \leq \mathbf{p}, \quad \mathbf{z} \geq \mathbf{0} \tag{4.2}$$

$$\mathbf{x} \in \text{argmin}_{\mathbf{x}'}\{\, \boldsymbol{\psi} \cdot \mathbf{x}' : A\mathbf{x}' \leq B\mathbf{z} + \mathbf{u}, \quad \mathbf{x}' \in \mathbb{Z}^n \,\}. \tag{4.3}$$

All data are also assumed to be integer.

Here, the set of *bilevel feasible solutions* is given as:

$$\mathcal{F} = \{(\mathbf{x}, \mathbf{z}) \in P : \mathbf{x} \in \text{argmin}\{\, \boldsymbol{\psi} \cdot \mathbf{x}' : A\mathbf{x}' \leq B\mathbf{z} + \mathbf{u}, \, \mathbf{x} \in \mathbb{Z}^n \,\}\}. \tag{4.4}$$

We assume that this *bilevel feasible set* $\mathcal{F}$ is bounded (though possibly empty). As discussed in the introductory chapter, $\mathcal{F}$ is known to be nonconvex in general, and difficult to describe directly [18]. Moreover, given $\mathbf{z}$ is continuous it is possible that $\mathcal{F}$ is not closed

[83]. This possibility is reflected in our definition of BMILP in (4.1)–(4.3) where the leader's optimization is expressed as an infimum and not a minimum. When $\mathbf{z}$ is also integer and the instance is feasible, then an optimal solution always exists and "inf" can be replaced with "min" [83].

Example 4.1.1. To demonstrate this complication, consider the following simple example.

$$\begin{aligned} \inf_{x,z} \quad & -x + z \\ \text{s.t.} \quad & 0 \leq z \leq 1 \\ & x \in \operatorname{argmin}_{x'} \{\, x' : x' \geq z,\ 0 \leq x' \leq 1, \quad x' \in \mathbb{Z} \,\} \end{aligned}$$

The "argmin" then simply requires that $x = \lceil z \rceil$ and thus the problem reduces to:

$$\inf_{z} \{ z - \lceil z \rceil : 0 \leq z \leq 1 \}.$$

Clearly, the infimum is -1 and cannot be attained. Observe that the set of bilevel feasible solutions

$$\mathcal{F} = \{(0,0)\} \bigcup (\{1\} \times (0,1])$$

is neither convex nor closed.

The approaches to problem (4.1)–(4.3) in the literature use traditional ideas from integer optimization, such as branch-and-bound enumeration [61] or cutting planes [32], all based on continuous relaxations. It is well documented that there are numerous pitfalls when applying standard ideas of branch-and-bound or cutting plane methods to a bilevel setting (see [32] for a discussion). Moreover, the algorithm introduced by Moore and Bard in [61] applies only to the case that both $\mathbf{x}$ and $\mathbf{z}$ are integer, or otherwise where it is known that an optimum solution exists. Thus, in general, this method cannot solve (4.1)–(4.3), in the sense that it cannot distinguish when an infimum is attained or not. The algorithm of DeNegre and

Ralphs [32] deals exclusively with the pure integer setting. In addition, these two papers give no run-time guarantees.

This chapter makes the following contributions:

(i) We present (Theorem 4.3.1) an algorithm which solves BMILP (4.1)–(4.3) in full generality (under our boundedness and integral data assumptions) by identifying when the infimum is attained and finding an optimal solution if one exists. No previously proposed algorithm is capable of this.

(ii) Our algorithms run in polynomial time when the follower's dimension n is fixed for the mixed integer case (i.e., z is continuous), and when the total dimension $n+d$ is fixed in the pure integer case (i.e., when z is integer). To the authors' knowledge, these are the first polynomial run-time guarantees of this type for discrete bilevel optimization. These results can be seen as a common extension of two important results on optimization in fixed dimension. The first is due to Lenstra [58], who establishes a polynomial time algorithm for integer programming with a fixed number of variables. The second is due to Deng [33], who develops a polynomial time algorithm for bilevel *linear* programs when the number of decision variables of the follower is fixed. Our result extends Lenstra's result to a bilevel setting and Deng's result to an integer setting. For further discussion of complexity issues in bilevel optimization see [17].

(iii) When no optimal solution to BMILP exists, we find, for any given $\epsilon > 0$ and in time polynomial in the input size and $\log(1/\epsilon)$, a solution with objective value no more than $1+\epsilon$ times the infimum.

(iv) We consider a previously unexplored generalization of bilevel integer programs where the follower has r linear objectives representing competing interests. Understanding problems with multiple criteria and multiple levels has been identified as an impor-

tant problem in the field of bilevel optimization [17]. We provide a polynomial time algorithm to find optimal solutions when the dimension $n + d$ is fixed (Theorem 4.5.1).

In Section 4.2, we introduce formal definitions and prove our first key result: when an instance of BMILP is feasible its infimum value is a rational number with polynomially bounded binary encoding size. In Section 4.3, we apply recent results from the study of parametric integer programming by Eisenbrand and Shmonin [37], based on the work of Kannan [49]. These references provide polynomial time algorithms for decision versions of parametric integer programs. We apply these algorithms in the mixed integer bilevel setting and, using binary search, derive our main result (Theorem 4.3.1): a polynomial time algorithm for the mixed integer case where $\mathbf{z}$ is continuous and the follower's dimension n is fixed. Finally, in Section 4.4 we present an algorithm using similar ideas to find optimal solutions in the pure integer case; that is, BMILP under the further restriction that $\mathbf{z}$ is integer. The algorithm runs in polynomial time when the total dimension $n + d$ is fixed.

In Section 4.5 we also consider two generalizations of (4.1)–(4.3) when $\mathbf{z}$ is assumed to be integer and analyze them using generating functions. The first generalization is to a special case of nonlinear discrete bilevel optimization. Such problems have been scarcely studied in the literature, with only very few references [43, 46, 72]. These references either treat special cases where the problems have very special structure (such as separable or monotone objectives) or are solved heuristically. Gümüş and Fluodas [43] provide an ϵ-approximation algorithm for a class of nonlinear discrete bilevel optimization problems by converting them to large global continuous optimization problems which are then solved to ϵ-optimality. We describe a fully polynomial time approximation scheme (FPTAS) for the setting where the leader's objective f is a nonnegative, not necessarily convex, polynomial, thus establishing a polynomial run-time guarantee in this special setting.

The second generalization is to a multi-objective bilevel integer programming problem where the follower has r linear objectives representing competing interests. An optimal

solution of such a problem is defined to be a feasible solution where the follower's action $\mathbf{x}$ a Pareto efficient outcome in the resulting lower level problem where the leader chooses $\mathbf{z}$ so as to optimize a linear objective $\mathbf{c} \cdot \mathbf{x} + \mathbf{e} \cdot \mathbf{z}$. This formulation extends the "optimistic assumption" to a multi-objective setting where we assume the leader can choose any $\mathbf{x}$ from the Pareto efficient set. We provide a polynomial time algorithm to find optimal solutions when the dimension $n + d$ is fixed (Theorem 4.5.1). This setting has not been explored in the literature. The case where the leader has multiple objectives was briefly explored in [31], but our methods do not apply to that setting.

4.2 Definition and preliminary results

An instance of the bilevel mixed integer linear program (BMILP) (4.1)–(4.3) is defined by the leader's and follower's dimensions d and n; the numbers h and m of linear inequality constraints in (4.2) and in the feasible set of the follower's subproblem (4.3); integer matrices $A \in \mathbb{Z}^{m\times n}$, $B \in \mathbb{Z}^{m\times d}$, $C \in \mathbb{Z}^{h\times n}$ and $D \in \mathbb{Z}^{h\times d}$; integer objective coefficient vectors $\mathbf{c} \in \mathbb{Z}^n$, $\mathbf{e} \in \mathbb{Z}^d$ and $\boldsymbol{\psi} \in \mathbb{Z}^n$; and integer right-hand side vectors $\mathbf{u} \in \mathbb{Z}^m$ and $\mathbf{p} \in \mathbb{Z}^h$. The binary encoding input size of an instance is the total number of bits needed to input all the data defining the instance.

We assume that the upper level polyhedron

$$P = \{(\mathbf{x}, \mathbf{z}) \in \mathbb{R}^{n+d} : (4.2)\} \tag{4.5}$$

is bounded (but possibly empty). We also assume that for every $\mathbf{z} \in \mathrm{proj}_{\mathbf{z}} P$, where

$$\mathrm{proj}_{\mathbf{z}} P = \{\mathbf{z} \in \mathbb{R}^d : (\mathbf{x}, \mathbf{z}) \in P \text{ for some } \mathbf{x} \in \mathbb{R}^n\},$$

the lower level feasible set $\{(\mathbf{x}, \mathbf{z}) \in \mathbb{R}^{n+d} : A\mathbf{x} \leq B\mathbf{z} + \mathbf{u}\}$ is also bounded (and possibly

empty). (For example, by Farkas's Lemma, if lower bounds $\mathbf{x} \geq \underline{\mathbf{x}}$ and $\mathbf{z} \geq \underline{\mathbf{z}}$ are part of or implied by the constraints defining P, and also by those defining the lower level feasible set, then the boundedness assumptions are satisfied if and only if the $(n+d)$-row vector of all 1's is a nonnegative combination of the rows of the constraint matrix (C, D), and also a nonnegative combination of the rows of the matrix $(A, -B)$.)

Under these boundedness assumptions, an instance of BMILP is either infeasible or else must have a bounded infimum value. Feasibility can be decided in polynomial time by applying Lenstra's (mixed) integer programming algorithm [58] to decide if $\{(\mathbf{x}, \mathbf{z}) \in P : A\mathbf{x} \leq B\mathbf{z} + \mathbf{u}, \quad \mathbf{x} \in \mathbb{Z}^n\} \neq \emptyset$. A key result used in the sequel is that, when the instance of BMILP is feasible, its infimum value is a rational number with polynomially-bounded binary encoding size.

Proposition 4.2.1. *Assume that a given instance of problem* (4.1)–(4.3) *is feasible and has a bounded optimum (infimum) value. Then the infimum is a rational number with denominator bounded above by the maximum absolute value of a sub-determinant of the matrix* $(D, B)^T$.

Proof. Let $(\bar{\mathbf{x}}, \bar{\mathbf{z}}) \in \mathcal{F}$ be a bilevel feasible solution to (4.1)–(4.3) and define

$$Q(\bar{\mathbf{x}}, \bar{\mathbf{z}}) = \{\mathbf{z} \in \mathbb{R}^d : D\mathbf{z} \leq \mathbf{p} - C\bar{\mathbf{x}}; \quad \beta_i(\bar{\mathbf{z}}) \leq B_i\mathbf{z} + u_i < \beta_i(\bar{\mathbf{z}}) + 1 \ \forall i \in M\},$$

where $M = \{1, \ldots, m\}$, B_i is the i-th row of B, and $\beta_i(\bar{\mathbf{z}}) = \lfloor B_i\bar{\mathbf{z}} + u_i \rfloor$. Note that $\bar{\mathbf{z}} \in Q(\bar{\mathbf{x}}, \bar{\mathbf{z}})$, so $Q(\bar{\mathbf{x}}, \bar{\mathbf{z}})$ is a nonempty, bounded quasi-polyhedral set, i.e., its topological closure is

$$\mathrm{cl}\, Q(\bar{\mathbf{x}}, \bar{\mathbf{z}}) = \{\mathbf{z} \in \mathbb{R}^d : D\mathbf{z} \leq \mathbf{p} - C\bar{\mathbf{x}}; \quad \beta_i(\bar{\mathbf{z}}) \leq B_i\mathbf{z} + u_i \leq \beta_i(\bar{\mathbf{z}}) + 1 \ \forall i \in M\}, \tag{4.6}$$

a nonempty polytope. Every $\mathbf{z} \in Q(\bar{\mathbf{x}}, \bar{\mathbf{z}})$ satisfies $\lfloor B_i\mathbf{z} + u_i \rfloor = \lfloor B_i\bar{\mathbf{z}} + u_i \rfloor$ for all $i = 1, \ldots, m$

and, since A is integer,

$$\forall \mathbf{x} \in \mathbb{Z}^n : \quad A\mathbf{x} \le B\mathbf{z} + \mathbf{u} \Longleftrightarrow A\mathbf{x} \le B\bar{\mathbf{z}} + \mathbf{u},$$

implying that $\bar{\mathbf{x}} \in \operatorname{argmin}\{\boldsymbol{\psi} \cdot \mathbf{x} : A\mathbf{x} \le B\mathbf{z} + \mathbf{u},\ \mathbf{x} \in \mathbb{Z}^n\}$ for all $\mathbf{z} \in Q(\bar{\mathbf{x}}, \bar{\mathbf{z}})$, that is,

$$\forall \mathbf{z} \in Q(\bar{\mathbf{x}}, \bar{\mathbf{z}}) \quad (\bar{\mathbf{x}}, \mathbf{z}) \in \mathcal{F}. \tag{4.7}$$

Now let v^* denote the infimum value in BMILP (4.1)–(4.3). There is a sequence $(\hat{\mathbf{x}}^k, \hat{\mathbf{z}}^k)_{k\in\mathbb{N}}$ of bilevel feasible solutions with objective value $\mathbf{c} \cdot \hat{\mathbf{x}}^k + \mathbf{e} \cdot \hat{\mathbf{z}}^k$ converging to v^*. Since P is a polytope, its projection $\operatorname{proj}_{\mathbf{x}} P = \{\mathbf{x} \in \mathbb{R}^n : (\mathbf{x}, \mathbf{z}) \in P \text{ for some } \mathbf{z} \in \mathbb{R}^d\}$ contains a finite number of integer points $\mathbf{x} \in P \cap \mathbb{Z}^n$. Therefore there is a subsequence $(\tilde{\mathbf{x}}^k, \tilde{\mathbf{z}}^k)_{k\in\mathbb{N}}$ with all $\tilde{\mathbf{x}}^k = \tilde{\mathbf{x}}^0$. For every $i \in M$ there is also a finite number of possible integer values $\beta_i(\mathbf{z}) = \lfloor B_i\mathbf{z} + u_i \rfloor$ when $\mathbf{z} \in \operatorname{proj}_{\mathbf{z}} P$. Thus there exists a subsequence $(\tilde{\mathbf{x}}^0, \breve{\mathbf{z}}^k)_{k\in\mathbb{N}}$ in $\mathcal{F}$ with all $\beta(\breve{\mathbf{z}}^k) = \beta(\breve{\mathbf{z}}^0)$, i.e., $(\tilde{\mathbf{x}}^0, \breve{\mathbf{z}}^k) \in Q(\tilde{\mathbf{x}}^0, \breve{\mathbf{z}}^0)$, and $\mathbf{c}\cdot\tilde{\mathbf{x}}^0 + \mathbf{e}\cdot\breve{\mathbf{z}}^k$ converging to v^*. From (4.7) it follows that $v^* = \mathbf{c}\cdot\tilde{\mathbf{x}}^0 + \inf_{\mathbf{z}}\{\mathbf{e}\cdot\mathbf{z} : \mathbf{z} \in Q(\tilde{\mathbf{x}}^0, \breve{\mathbf{z}}^0)\}$. By continuity of the objective $\mathbf{e}\cdot\mathbf{z}$, $v^* - \mathbf{c}\cdot\tilde{\mathbf{x}}^0$ is the optimum value of the linear programming problem $\min_{\mathbf{z}}\{\mathbf{e}\cdot\mathbf{z} : \mathbf{z} \in \operatorname{cl} Q(\tilde{\mathbf{x}}^0, \breve{\mathbf{z}}^0)\}$. By (4.6), for every $i \in M$ at most one of the two constraints $\beta_i(\breve{\mathbf{z}}^0) \le B_i\mathbf{z} + u_i$ or $B_i\mathbf{z} + u_i \le \beta_i(\breve{\mathbf{z}}^0) + 1$ is binding at the LP optimum, hence $v^* - \mathbf{c}\cdot\tilde{\mathbf{x}}^0$ must also be the optimum value of a linear programming problem

$$\begin{aligned} \min \quad & \mathbf{e}\cdot\mathbf{z} \\ \text{s.t.} \quad & D\mathbf{z} \le \mathbf{p} - C\tilde{\mathbf{x}}^0 \\ & -B_i\mathbf{z} \le u_i - \beta_i(\breve{\mathbf{z}}^0), \qquad \forall i \in I \\ & B_j\mathbf{z} \le -u_j + \beta_j(\breve{\mathbf{z}}^0) + 1, \quad \forall j \in J \end{aligned} \tag{4.8}$$

for some disjoint $I,\ J \subseteq M$. It is well known that the optimal value of a bounded linear

program with integer data occurs at a basic feasible solution, the denominators of which are bounded by the maximum absolute value of a sub-determinant of its constraint matrix. By simple properties of determinants the stated result then follows. □

Note that in the pure integer case, where the leader's variables are also required to be integer, the bilevel feasible set is

$$\mathcal{F}' = \left\{(\mathbf{x}, \mathbf{z}) \in P : \mathbf{z} \in \mathbb{Z}^d, \;\; \mathbf{x} \in \operatorname{argmin}\{\, \boldsymbol{\psi} \cdot \mathbf{x}' : A\mathbf{x}' \leq B\mathbf{z} + \mathbf{u},\, \mathbf{x}' \in \mathbb{Z}^n \,\}\right\}. \tag{4.9}$$

When it is non-empty, $\mathcal{F}'$ is a finite set, and then the infimum in (4.1) is attained. Since $\mathbf{c}$ and $\mathbf{e}$ are integral, this optimum value is integral, providing a trivial strengthening of Proposition 4.2.1 in the pure integer case.

4.3 An algorithm for bilevel mixed integer linear programming

In this section we prove the main result of this chapter:

Theorem 4.3.1. *There exists an algorithm which solves the* BMILP *problem* (4.1)–(4.3) *in the following sense: the algorithm*

(i) decides if the instance is feasible;

(ii) if it is feasible, decides if an optimal solution exists (that is, if the infimum is attained); and

(iii) if the infimum is attained, finds an optimal solution.

The algorithm runs in polynomial time when the follower's dimension n is fixed.

To prove this theorem, we use results on parametric integer programs with parameterized right-hand sides. Parametric integer programming is well-studied, the particular approach we discuss builds on geometric and algorithmic number theory and has it roots in the work of Lenstra [58] on integer linear programming in fixed dimension. We draw mainly on the recent work of Eisenbrand and Shmonin [37], which itself builds on earlier results by Kannan [49]. These two related works solve a decision version of parametric integer programming (defined below) in polynomial time when certain parameters are fixed. As discussed in Section 4.1, we can view the lower level problem as a parametric integer program. But the connection between parametric integer programming and bilevel integer programming in fact goes deeper, as we now discuss.

The *integer projection* of a polyhedron $Q \subseteq \mathbb{R}^{m+p}$ is the set

$$Q/\mathbb{Z}^p = \{\mathbf{b} \in \mathbb{R}^m : (\mathbf{b}, \mathbf{w}) \in Q \text{ for some } \mathbf{w} \in \mathbb{Z}^p\}. \tag{4.10}$$

The feasibility version of parametric integer linear programming is the following decision problem:

PILP: Given a rational matrix $R \in \mathbb{Q}^{m\times n}$ and a rational polyhedron $Q \in \mathbb{R}^{m+p}$ decide if the linear system $R\mathbf{y} \leq \mathbf{b}$ has an integer solution for all $\mathbf{b} \in Q/\mathbb{Z}^p$.

Letting

$$P_{\mathbf{b}} = \{\mathbf{y} \in \mathbb{R}^n : R\mathbf{y} \leq \mathbf{b}\},$$

this problem is equivalent, by negation, to deciding the sentence

$$\exists \mathbf{b} \in Q/\mathbb{Z}^p, \quad P_{\mathbf{b}} \cap \mathbb{Z}^n = \emptyset. \tag{4.11}$$

Eisenbrand and Shmonin [37] prove:

Theorem 4.3.2. (Theorem 4.2 in [37]) *There is an algorithm that, given a rational matrix $R \in \mathbb{Q}^{m\times n}$ and a rational polyhedron $Q \subset \mathbb{R}^{m+p}$, decides the sentence* (4.11)*. The algorithm runs in polynomial time if p and n are fixed.*

Our goal is to establish a connection between PILP and BMILP so this result can be used in solving BMILP. First we define a decision version of our problem:

> BMILP$_\alpha$: Given integer matrices $A \in \mathbb{Z}^{m\times n}$, $B \in \mathbb{Z}^{m\times d}$, $C \in \mathbb{Z}^{h\times n}$ and $D \in \mathbb{Z}^{h\times d}$; integer vectors $\mathbf{c} \in \mathbb{Z}^n$, $\mathbf{e} \in \mathbb{Z}^d$, $\boldsymbol{\psi} \in \mathbb{Z}^n$, $\mathbf{u} \in \mathbb{Z}^m$ and $\mathbf{p} \in \mathbb{Z}^h$; and rational scalar $\alpha \in \mathbb{Q}$, decide if there exists a vector $(\mathbf{x}, \mathbf{z}) \in \mathbb{R}^{n+d}$ such that (4.2), (4.3) and $\mathbf{c}\cdot\mathbf{x} + \mathbf{e}\cdot\mathbf{z} \leq \alpha$.

Proposition 4.3.3. *There exists an algorithm that decides* BMILP$_\alpha$*. The algorithm runs in polynomial time when the follower's dimension n is fixed.*

Proof. Since $\boldsymbol{\psi}$ is integral, we can express (4.3) as the fact that

$$\{\mathbf{x}' \in \mathbb{R}^n : \boldsymbol{\psi}\cdot\mathbf{x}' \leq \boldsymbol{\psi}\cdot\mathbf{x} - 1, \quad A\mathbf{x}' \leq B\mathbf{z} + \mathbf{u}\} \cap \mathbb{Z}^n = \emptyset. \tag{4.12}$$

Let $R = (\boldsymbol{\psi}, A, 0_{d\times n})^T$ where $0_{d\times n}$ is a $d \times n$ matrix of zeroes; $\mathbf{b} = (b_{(1)}, \mathbf{b}_{(2)}, \mathbf{b}_{(3)})^T$ with $b_{(1)} = \boldsymbol{\psi}\cdot\mathbf{x} - 1$, $\mathbf{b}_{(2)} = B\mathbf{z} + \mathbf{u}$ and $\mathbf{b}_{(3)} = \mathbf{z}$. Then (4.3) and $\mathbf{z} \geq \mathbf{0}$ together are equivalent to $P'_{\mathbf{b}} \cap \mathbb{Z}^n = \emptyset$ where $P'_{\mathbf{b}} = \{\mathbf{x}' \in \mathbb{R}^n : R\mathbf{x}' \leq \mathbf{b}\}$. Letting

$$\begin{aligned} Q_\alpha = \{(\mathbf{b}, \mathbf{x}) \quad : \quad & \mathbf{c}\cdot\mathbf{x} + \mathbf{e}\cdot\mathbf{b}_{(3)} \leq \alpha, \quad C\mathbf{x} + D\mathbf{b}_{(3)} \leq \mathbf{p}, \quad \mathbf{b}_{(3)} \geq \mathbf{0}, \\ & A\mathbf{x} \leq \mathbf{b}_{(2)}, \quad b_{(1)} = \boldsymbol{\psi}\cdot\mathbf{x} - 1, \quad \mathbf{b}_{(2)} = B\mathbf{b}_{(3)} + \mathbf{u}\}, \end{aligned} \tag{4.13}$$

deciding BMILP$_\alpha$ reduces to deciding an instance of (4.11) with $p = n$, $\mathbf{w} = \mathbf{x}$, $Q = Q_\alpha$ and $P_{\mathbf{b}} = P'_{\mathbf{b}}$. The result now follows from Theorem 4.3.2. □

Our approach to the proof of Theorem 4.3.1 is to use a binary search procedure for target

value α in BMILP$_\alpha$ and related decision problems, so as to find the infimum value of problem BMILP and an optimal solution if one exists.

The following result from Kwek and Mehlhorn [55] yields the binary search algorithm used in proving our results:

Theorem 4.3.4. (Theorem 1 in [55]) *There exists a $\theta(\log L)$-time algorithm that determines an unknown rational number r which lies in the set $\mathcal{Q}_L = \left\{\frac{p}{q} : p, q \in \{1, \ldots, L\}\right\}$ by asking at most $2\log_2 L + O(1)$ queries of the form "is $r \leq \alpha$?".*

We first establish:

Lemma 4.3.5. *Consider a feasible instance to* BMILP *problem* (4.1)–(4.3). *Its infimum value can be determined in polynomial time when the dimension n is fixed.*

Proof. We first determine polynomial-sized lower and upper bounds $\underline{v}$ and $\bar{v}$ on the infimum value v^* by solving in polynomial time the linear programs $\underline{v} = \min\{\mathbf{c}\cdot\mathbf{x} + \mathbf{e}\cdot\mathbf{z} : (\mathbf{x}, \mathbf{z}) \in P\}$ and $\bar{v} = \max\{\mathbf{c}\cdot\mathbf{x} + \mathbf{e}\cdot\mathbf{z} : (\mathbf{x}, \mathbf{z}) \in P\}$. We then perform ordinary binary search for v^* in successive intervals $[v', v'']$ satisfying $v' < v^* \leq v''$, where the search query "is $v^* \leq \alpha$" is the decision problem BMILP$_\alpha$, and the initial interval $[v', v''] = [\underline{v} - 1, \bar{v}]$. We stop, after a polynomial number of queries, as soon as $v'' - v' < 1$. If there exists an integer β such that $v' < \beta < v''$ then we perform one more query "is $v^* \leq \beta$" and set the current interval to $[\beta - 1, \beta]$ if the response was positive (i.e., if $v^* \leq \beta$), and to $[\beta, v'']$ otherwise. If there is no such integer β then we set the current interval to $[\lfloor v' \rfloor, v'']$. In any case we obtain an interval $[\bar{v}', \bar{v}'']$ such that $\bar{v}' < v^* \leq \bar{v}''$, $\bar{v}'' - \bar{v}' \leq 1$ and $\bar{v}'$ is integer. We now invoke the binary search algorithm of Theorem 4.3.4 for the rational number $r = v^* - \bar{v}' \in \mathcal{Q}_L$ where L is the upper bound on the denominator of v^* from Proposition 4.2.1. □

Example 4.3.6. To illustrate the lemma and visualize some of the concepts we consider the

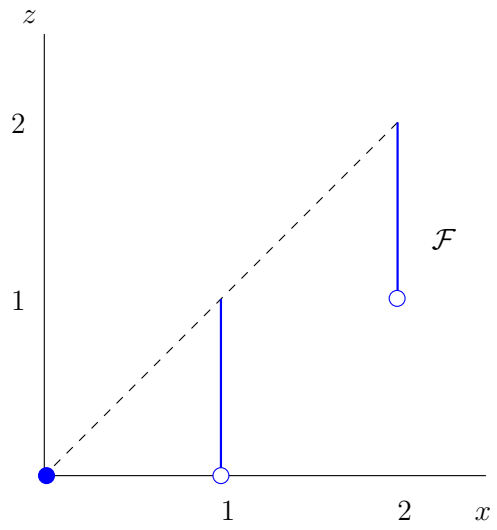


Figure 4.1: Bilevel feasible region for Example 4.3.6

following a simple BMILP:

$$\begin{aligned} \inf_{x,z} \quad & -3x + z \\ \text{s. t.} \quad & 0 \le x \le 2,\ 0 \le z \le 2 \\ & \mathbf{x} \in \operatorname{argmin}\{\, -x' : x \ge z,\ x' \in \mathbb{Z}\,\}. \end{aligned}$$

The bilevel feasible region $\mathcal{F}$ is shown in Figure 4.3.6. To illustrate the lemma first note that we can bound the denominator of the infimum value v^* by 1. This follows from Proposition 4.2.1 and the fact $(D, B)^T = (1, -1)^T$. Thus v^* is an integer. To set the lower and upper bounds for our binary search we note that $\underline{v} = -6$ which is attained at the point $(2, 0)$ and $\bar{v} = 2$ which is attained at $(0, 2)$. The binary search starts with a query at the midpoint of the interval $[\underline{v}, \bar{v}]$, namely -2. A call to BMILP$_{-2}$ returns true since, for instance, $(x, y) = (1, 1) \in \mathcal{F}$ satisfies $-3x + z \le -2$. We restrict the search for v^* to the interval $[-6, -2]$. Again we test the midpoint, and noting that BMILP$_{-4}$ returns true, restrict our search to the interval $[-6, -4]$. Now the midpoint of the search region is -5 and BMILP$_{-5}$ returns false. After another iteration the search region becomes $[-5, -4.5]$. We conclude

that $v^* = 5$, since it is the only integer remaining in search region.

We are now ready to prove Theorem 4.3.1.

Proof of Theorem 4.3.1. (i) As indicated in Section 4.2, feasibility can be decided in polynomial time by applying Lenstra's (mixed) integer programming algorithm [58] to decide if $\{(\mathbf{x}, \mathbf{z}) \in P : A\mathbf{x} \leq B\mathbf{z} + \mathbf{u}, \quad \mathbf{x} \in \mathbb{Z}^n\} \neq \emptyset$.

(ii) Assume the instance is feasible. By Lemma 4.3.5 we can find the infimum value v^* in polynomial time. This infimum is attained if and only if there exists a bilevel feasible $(\mathbf{x}, \mathbf{z}) \in \mathcal{F}$ such that $\mathbf{c} \cdot \mathbf{x} + \mathbf{e} \cdot \mathbf{z} = v^*$. This can be decided in polynomial time as in the proof of Proposition 4.3.3 by replacing the first constraint in the definition of Q_α in equation (4.13) with $\mathbf{c} \cdot \mathbf{x} + \mathbf{e} \cdot \mathbf{b}_{(3)} = v^*$.

(iii) Finally, assume that the infimum value is attained. We extend to bilevel mixed integer programming a standard method used in (ordinary) mixed integer programming to successively determine an optimum solution given an optimum objective value oracle (see for instance [74], page 260). In this method, we first find the optimum value of related instances of BMILP to successively determine each component of an optimal solution $(\mathbf{x}^*, \mathbf{z}^*)$ with lexicographically smallest integer component $\mathbf{x}^*$. By induction, assume that all components x_k^* for $k = 1, \ldots, j-1$ have already been determined. Then x_j^* is the optimum value of the BMILP

$$\begin{array}{ll} \min\limits_{\mathbf{x},\mathbf{z}} & x_j \\ \text{s. t.} & C\mathbf{x} + D\mathbf{z} \leq \mathbf{p}; \quad \mathbf{z} \geq \mathbf{0}; \quad \mathbf{c} \cdot \mathbf{x} + \mathbf{e} \cdot \mathbf{z} = v^*; \\ & x_k = x_k^* \text{ for } k = 1, \ldots, j-1; \quad (4.3). \end{array}$$

Since this problem is feasible and its objective is integer valued and bounded, its optimum value is attained, and is determined in polynomial time by Lemma 4.3.5. (Note that, since

this optimal value x_j^* is integer, we only need to use ordinary binary search and stop when the width of the current interval is less than one.)

Having determined $\mathbf{x}^*$, we next determine the continuous component $\mathbf{z}^*$. Note that, even though $\mathbf{x}^*$ has now been fixed, $\mathbf{z}^*$ cannot be directly found by solving a simple linear program, because we still need to enforce the (nonconvex) incentive compatibility constraint (4.3). Using ideas from the proof of Proposition 4.2.1, we construct an "integer RHS vector" $\mathbf{r} = (\lfloor B_i\mathbf{z} + u_i \rfloor)_{i=1,\dots m}$ which is lexicographically minimum among all those defined by optimum BMILP solutions $(\mathbf{x}^*, \mathbf{z})$. Thus by induction assume that for $k = 1, \dots, i-1$ we have determined integers r_k such that the set

$$R_{i-1} = \left\{ (\mathbf{x}, \mathbf{z}) \in \mathbb{R}^{n+d} : \begin{array}{l} C\mathbf{x}^* + D\mathbf{z} \le \mathbf{p}; \quad \mathbf{z} \ge \mathbf{0}; \quad \mathbf{c}\cdot\mathbf{x}^* + \mathbf{e}\cdot\mathbf{z} = v^*; \\ r_k \le B_k\,\mathbf{z} + u_k < r_k + 1 \text{ for all } k < i; \quad \mathbf{x} = \mathbf{x}^*; \end{array} \quad (4.3) \right\}$$

is nonempty. As in the proof of Proposition 4.2.1, note that the closure $\operatorname{cl} R_{i-1}$ is determined by replacing each strict inequality $B_k\mathbf{z} + u_k < r_k + 1$ with $B_k\mathbf{z} + u_k \le r_k + 1$, and is thus a nonempty polytope. We then compute the infimum value of the BMILP

$$\rho_i = \inf_{\mathbf{x},\mathbf{z}} \{ B_i\,\mathbf{z} + u_i : \mathbf{z} \in \operatorname{cl} R_{i-1} \}, \tag{4.14}$$

in polynomial time by Lemma 4.3.5. Let $r_i = \lfloor \rho_i \rfloor$. We claim that $R_i \neq \emptyset$. To prove this claim, let $\big((\mathbf{x}^*, \bar{\mathbf{z}}^j) \in R_{i-1}\big)_{j\in\mathbb{N}}$ be a sequence of points in $\operatorname{cl} R_{i-1}$ with $\lim_{j\to\infty}(B_i\,\bar{\mathbf{z}}^j + u_i) = \rho_i$. By continuity of the linear function $(\mathbf{x}^*, \mathbf{z}) \mapsto B_i\,\mathbf{z} + u_i$, there exists a sequence $\big((\mathbf{x}^*, \mathbf{z}^j) \in R_{i-1}\big)_{j\in\mathbb{N}}$ of points in R_{i-1} with $\lim_{j\to\infty}(B_i\,\mathbf{z}^j + u_i) = \rho_i < \lfloor \rho_i \rfloor + 1$. This implies that $\lfloor \rho_i \rfloor \le B_i\,\mathbf{z}^j + u_i < \lfloor \rho_i \rfloor + 1$ for some $j \in \mathbb{N}$, and therefore $(\mathbf{x}^*, \mathbf{z}^j) \in R_i$, showing that $R_i \neq \emptyset$.

Thus, after determining $\mathbf{x}^*$ and computing the infimum value ρ_i of m BMILPs (4.14), we obtain the polynomial-sized, integer vector $\mathbf{r} = \big(\lfloor \rho_i \rfloor\big)_{i=1,\dots,m}$ such that there exists an

optimal solution $(\mathbf{x}^*, \mathbf{z})$ to the original BMILP (4.1)–(4.3) satisfying

$$\mathbf{z} \in \operatorname{proj}_{\mathbf{z}} R_m \subseteq Q := \left\{ \mathbf{z} \in \mathbb{R}^d : \begin{array}{l} C\mathbf{x}^* + D\mathbf{z} \leq \mathbf{p}, \quad \mathbf{z} \geq \mathbf{0}, \quad \mathbf{c} \cdot \mathbf{x}^* + \mathbf{e} \cdot \mathbf{z} = v^* \\ \mathbf{r} \leq B\mathbf{z} + \mathbf{u} < \mathbf{r} + \mathbf{1} \end{array} \right\}$$

where $\mathbf{1}$ is the all-ones vector in $\mathbb{Z}^m$. As in the proof of Proposition 4.2.1 we note that every $\mathbf{z} \in Q$ gives rise to an optimal solution $(\mathbf{x}^*, \mathbf{z})$. We now show how to find one such optimal solution whose binary encoding length is polynomial in the input size. Let $k = 1 + \dim Q$. Find k affinely independent vertices $\mathbf{z}^1, \ldots, \mathbf{z}^k$ of $\operatorname{cl} Q$. This can be done in polynomial time since $\operatorname{cl} Q$ is a polytope (see, e.g., [74], page 186). By standard results in linear optimization, the vertices $\mathbf{z}^1, \ldots, \mathbf{z}^k$ can be written as rational numbers in $\frac{1}{\Delta}\mathbb{Z}^d$ where Δ is the least common multiple of the determinants of the bases of the constraint submatrix associated with the basic feasible solutions $\mathbf{z}^1, \ldots, \mathbf{z}^k$. Note that Δ has encoding length polynomial in the input size. It is possible that none of the $\mathbf{z}^j$ lie in Q, since Q is not closed, but nonetheless their barycenter $\mathbf{z}^* = \sum_{j=1}^k \frac{1}{k}\, \mathbf{z}^j$ is guaranteed to be in Q and also in the set $\frac{1}{k\Delta}\mathbb{Z}^d$. Thus it has a binary encoding length polynomial in the input size.

It follows that $(\mathbf{x}^*, \mathbf{z}^*)$ is an optimal solution to (4.1)–(4.3) and can be found in polynomial time. □

We note, in particular, that this theorem provides a polynomial-time algorithm to decide if a BMILP instance has an optimal solution. To the authors' knowledge, this is the first algorithm to achieve this in the literature on discrete bilevel optimization. The existence of solutions in the case where $\mathbf{z}$ is continuous has long been known not to be guaranteed (see for instance [61]) but little is known in terms of necessary and sufficient conditions for existence. Dempe and Richter [30] explore the setting where the lower level problem is a knapsack problem, and they provide necessary and sufficient conditions for existence of solutions, but there have been no studies of this kind in a more general setting. An efficient algorithm

to decide the existence of solutions, such as the one in Theorem 4.3.1, may be helpful in investigations of this kind.

A final question remains: what can we say about the BMILP problem (4.1)–(4.3) when the infimum is not attained? In such a case it may be desirable to find ϵ-optimal solutions whose objective values approximate the infimum. The following result demonstrates how we can adjust our arguments to find ϵ-optimal solutions.

Corollary 4.3.7. *Let $\epsilon > 0$ be given in binary encoding. Given a feasible instance of the* BMILP *problem* (4.1)–(4.3) *with infimum value v^*, there exists an algorithm to find a feasible solution $(\mathbf{x}, \mathbf{z})$ that satisfies $\mathbf{c} \cdot \mathbf{x} + \mathbf{e} \cdot \mathbf{z} \leq (1+\epsilon)v^*$. The algorithm runs in polynomial time when the follower's dimension n is fixed.*

Proof. By Theorem 4.3.1 part (ii) we can decide in polynomial time if the infimum is attained. If so, by using part (iii) of the same theorem we can find an optimal solution $(\mathbf{x}^*, \mathbf{z}^*)$ that satisfies $\mathbf{c} \cdot \mathbf{x}^* + \mathbf{e} \cdot \mathbf{z}^* = v^* \leq (1 + \epsilon)v^*$.

If we find that the infimum is not attained we apply a similar procedure as in the proof of Theorem 4.3.1 part (iii). By replacing the condition $\mathbf{c} \cdot \mathbf{x}^* + \mathbf{e} \cdot \mathbf{z} = v^*$ with the condition $\mathbf{c} \cdot \mathbf{x} + \mathbf{e} \cdot \mathbf{z} \leq (1 + \epsilon)v^*$, the result follows immediately. □

It should be noted that this algorithm achieves a stronger form of efficiency than a fully polynomial time approximation scheme (FPTAS). In an FPTAS, the error ϵ is given in unary encoding, whereas in our algorithm ϵ enters as input in binary encoding, since its value only plays a role in the constraint $\mathbf{c} \cdot \mathbf{x} + \mathbf{e} \cdot \mathbf{z} \leq (1 + \epsilon)v^*$. In this sense, we have obtained a "better than fully polynomial time" approximation scheme to yield solutions with objective value ϵ-close to the infimum.

A distinguishing feature of our main result, Theorem 4.3.1, is that the dimension d of the leader's variable need not be fixed in order to assure a polynomial running time. Note, however, that this result in no way conflicts with the fact that bilevel *linear* programming,

in which both $\mathbf{x}$ and $\mathbf{z}$ are continuous and the dimension of both $\mathbf{x}$ and $\mathbf{z}$ are allowed to vary, is an NP-hard problem. In our setting it is the leader's variable only which is allowed to be continuous and have varying dimension, and an important ingredient in our proof is that there are only finitely many integer right-hand sides $\beta_i(\mathbf{z}) = \lfloor B_i\mathbf{z} + \mathbf{u} \rfloor$ to consider when the matrix A and decision variables $\mathbf{x}$ are integer.

4.4 An algorithm for bilevel integer linear programming

We can also extend our results to the pure integer setting, where $\mathbf{z}$ is further restricted to be integer. We refer to this problem as the bilevel integer linear programming (BILP) problem, an instance of which is specified by the same data as BMILP and the same constraints (4.1)–(4.3) with the additional integrality condition

$$\mathbf{z} \in \mathbb{Z}^d. \tag{4.15}$$

As above, we assume that the feasible set $\mathcal{F}'$ defined in (4.9) is bounded.

Theorem 4.4.1. *There exists an algorithm that solves the* BILP *problem* (4.1)–(4.3) *&* (4.15) *in the following sense: the algorithm*

(i) decides if a given instance is feasible;

(ii) if it is feasible, finds an optimal solution.

The algorithm runs in polynomial time when the total dimension $d + n$ is fixed.

Proof. (i) Feasibility can be established in polynomial time by applying Lenstra's integer programming algorithm [58] to decide if $\{(\mathbf{x}, \mathbf{z}) \in P : A\mathbf{x} \leq B\mathbf{z} + \mathbf{u}, (\mathbf{x}, \mathbf{z}) \in \mathbb{Z}^{n+d}\} \neq \emptyset$.

(ii) Assume the instance is feasible. The idea of how to determine an optimal solution is similar to, but simpler than, the mixed integer case, as we use binary search to directly find the lexicographically minimal optimal solution $(\mathbf{x}^*, \mathbf{z}^*)$. To begin, consider the decision problem BILP_α, which is identical to BMILP_α with the additional integrality condition (4.15).

Problem BILP_α can be decided in polynomial time using an algorithm similar to that in Proposition 4.3.3. Indeed, if we simply redefine

$$\begin{aligned} Q_\alpha = \{(\mathbf{b}, \mathbf{x}, \mathbf{b}) \;\; : \;\; & \mathbf{c} \cdot \mathbf{x} + \mathbf{e} \cdot \mathbf{b}_{(3)} \leq \alpha, \;\; C\mathbf{x} + D\mathbf{b}_{(3)} \leq \mathbf{p}, \;\; \mathbf{b}_{(3)} \geq 0, \\ & A\mathbf{x} \leq \mathbf{b}_{(2)}, \;\; \mathbf{b}_{(1)} = \boldsymbol{\psi} \cdot \mathbf{x} - 1, \;\; \mathbf{b}_{(2)} = B\mathbf{b}_{(3)} + \mathbf{u}\}, \end{aligned}$$

and let $p = n + d$, the algorithm in the proof of Proposition 4.3.3 decides BILP_α.

It is then straightforward to find the optimal value v^* of the given BILP instance using binary search as in the proof of Lemma 4.3.5 and stopping when the search interval has length less than one.

Given the optimal value v^* we again use binary search to determine the lexicographically minimal optimal solution $(\mathbf{x}^*, \mathbf{z}^*)$. The procedure to find $\mathbf{x}^*$ is identical to that described in the proof of Theorem 4.3.1. The procedure to find $\mathbf{z}^*$ is also similar. We successively determine each component of $\mathbf{z}^*$ as follows: assume that all components z_k^* for $k = 1, \ldots, j-1$ have already been determined. Then z_j^* is the optimal value of the BILP

$$\begin{aligned} \min_{\mathbf{z}} \quad & z_j \\ \text{s. t.} \quad & C\mathbf{x}^* + D\mathbf{z} \leq \mathbf{p}; \;\; \mathbf{z} \geq 0; \;\; \mathbf{z} \in \mathbb{Z}^d \;\; \mathbf{c} \cdot \mathbf{x}^* + \mathbf{e} \cdot \mathbf{z} = v^*; \\ & z_k = z_k^* \;\; \text{for } k = 1, \ldots, j-1; \\ & \mathbf{x}^* \in \text{argmin}_{\mathbf{x}'} \{ \boldsymbol{\psi} \cdot \mathbf{x}' : A\mathbf{x}' \leq B\mathbf{z} + \mathbf{u}, \;\; \mathbf{x}' \in \mathbb{Z}^n \}. \end{aligned}$$

Since this problem is feasible and its objective is integer valued and bounded, its optimum

value is attained, and can be determined as in the previous paragraph.

The result then follows. □

4.5 Generating function approach

In the previous sections we were able to solve BMILP's via parametric integer programming results when the leader's decision $\mathbf{z}$ is either continuous or discrete using binary search. Here we again restrict $\mathbf{z}$ to be integer and consider an alternate approach to solving BILP using rational generating functions.

To be precise, we apply generating function methods to problem (4.2)–(4.3) & (4.15) where we replace the objective (4.1) with maximizing $f(\mathbf{x}, \mathbf{z})$ where f is either linear or a general nonnegative polynomial (not necessarily convex).

We first point that an algorithm for finding optimal solutions in the linear case (i.e. where f is linear) is very similar to that in the Stackelberg–Nash setting in Section 2.5. We mention only briefly here the differences: (i) there is only a single follower as opposed to a group of followers (hence any follower's optimal decision is trivially a "Nash equilibrium"); (ii) the payoffs are linear instead of piecewise linear; and (iii) the set constraining the leader's actions is more complicated and depends on the action of the follower, see (4.2), whereas in the Stackelberg–Nash setting in Section 2.5 the leader's choice set is independent of the actions of the followers. Nonetheless, these differences are minor and easily handled by the rational generating function technique described in Chapter 2. Details are thus omitted.

Given this encoding of the optimal solutions we can efficiently count and enumerate optimal solutions. These are straightforward applications of the Counting and the Enumeration Lemmas using the generating function of $\mathcal{F}^*$. Being able to count the number of bilevel feasible solutions is interesting because we can identify when there is a unique feasible solution. If this is the case, then we do not need to make the optimistic assumption and our

results are unambiguous. Note that no existing methods in the literature would be able to guarantee such information efficiently. To count all solutions by enumerative methods such as branch and bound would essentially involve an exhaustive search. Here, we can supply an exact count of the number of equilibria without resort to enumeration.

An FPTAS for the optimal value of f can be derived when using Lemma 1.4.11. Details are similar to results in previously chapters and thus omitted.

One final application of generating functions that is less straightforward is to multi-objective bilevel integer programming. We consider a setting where the follower has a fixed number r of competing linear objectives with cost vectors $\boldsymbol{\psi}_1, \ldots, \boldsymbol{\psi}_r$ and the leader has a single linear objective. As mentioned in the introduction, to the authors' knowledge this setting has not been considered in the literature, but is a simple extension of the approaches explored in this section. We can state the problem formally as:

$$\begin{aligned} \min \quad & \mathbf{c} \cdot \mathbf{x} + \mathbf{e} \cdot \mathbf{z} \\ \text{s.t.} \quad & (\mathbf{x}, \mathbf{z}) \in P \subseteq \mathbb{R}^{n+d}, \quad \mathbf{z} \in \mathbb{Z}^d \\ & \mathbf{x} \in \arg-\text{vmin}\{[\boldsymbol{\psi}_1 \cdot \mathbf{x}', \ldots, \boldsymbol{\psi}_r \cdot \mathbf{x}'] : A\mathbf{x}' \leq B\mathbf{z} + \mathbf{u},\ \mathbf{x}' \in \mathbb{N}^n\} \end{aligned} \tag{4.16}$$

where $\arg-\text{vmin}$ denotes the set of Pareto efficient vectors to the lower level multi-objective integer linear program:

$$\begin{aligned} \text{vmin} \quad & [\boldsymbol{\psi}_1 \cdot \mathbf{x}, \ldots, \boldsymbol{\psi}_r \cdot \mathbf{x}] \\ \text{s.t.} \quad & (\mathbf{x}, \mathbf{z}) \in P \subseteq \mathbb{R}^{n+d}, \quad \mathbf{z} \in \mathbb{Z}^d. \\ & A\mathbf{x} \leq B\mathbf{z} + \mathbf{u}, \mathbf{x} \in \mathbb{N}^n \end{aligned} \tag{4.17}$$

A feasible solution $\mathbf{x}$ to the lower level problem (4.17) is *Pareto efficient* if and only if there is no other point $\mathbf{x}'$ such that $\boldsymbol{\psi}_i \cdot \mathbf{x}' \leq \boldsymbol{\psi}_i \cdot \mathbf{x}'$ for all $i = 1, \ldots, r$ and $\boldsymbol{\psi}_j \cdot \mathbf{x}' \leq \boldsymbol{\psi}_j \cdot \mathbf{x}$ for some index j. Put in other words, when a feasible $\mathbf{x}$ is not Pareto efficient (that is, *Pareto dominated*) if there exists an $\mathbf{x}'$ such that $\psi_i \cdot x' \leq \psi_i \cdot x$ for all i and $\sum_{i=1}^{r} \psi_i \cdot \mathbf{x}' < \sum_{i=1}^{r} \psi_i \cdot \mathbf{x}$.

In this multi-objective bilevel integer programming problem the leader chooses $\mathbf{z}$, then the follower then chooses a feasible and Pareto efficient $\mathbf{x}$. In our formulation we have extended the "optimistic assumption" to this more general setting; that is, the leader has the ability to choose $\mathbf{x}$ from amongst the set of Pareto efficient responses of the follower as demonstrated in the fact that the leader can choose $\mathbf{x}$ from the $\arg - \text{vmin}$ set. Under this "optimistic assumption" we derive the following:

Theorem 4.5.1. *Fix the the dimension $n+d$. Then, there exists a polynomial time algorithm to solve the multi-objective bilevel integer program of the form* (4.16).

Proof. Let E denote the sets of pairs $(\mathbf{x}, \mathbf{z}) \in P$ where $\mathbf{x}$ is a Pareto efficient response to $\mathbf{z}$; that is,

$$E = \{(\mathbf{x}, \mathbf{z}) \in P : \mathbf{x} \in \arg - \text{vmin}\{[\boldsymbol{\psi}_1 \cdot \mathbf{x}', \ldots, \boldsymbol{\psi}_r \cdot \mathbf{x}'] : A\mathbf{x}' \leq B\mathbf{z} + \mathbf{u}, \mathbf{x}' \in \mathbb{N}^n\}$$

Let $\Omega = \{(\mathbf{x}, \mathbf{z}) \in P : A\mathbf{x} \leq \mathbf{B}\mathbf{z} + u\} \bigcap \mathbb{Z}^{n+d}$. It is straightforward to show that E can be described as follows:

$$E = \Omega \setminus D \tag{4.18}$$

where

$$D = \text{proj}_{(\mathbf{x},\mathbf{z})}\{(\mathbf{x}, \mathbf{z}, \mathbf{x}') : (\mathbf{x}, \mathbf{z}) \in \Omega, (\mathbf{x}', \mathbf{z}) \in \Omega,\ \psi_i \cdot x' \leq \psi_i \cdot x,\ \forall i;\ \sum_{i=1}^{r} \psi_i \cdot \mathbf{x}' \leq \sum_{i=1}^{r} \psi_i \cdot \mathbf{x} - 1\}$$

is the set of Pareto dominated solutions.

We use the Projection Lemma and Boolean Operations Lemma to derive a short rational generating function encoding of E. Now we apply the Linear Optimization Lemma (Lemma 1.4.10) using the generating function of E and the linear objective $\mathbf{c} \cdot \mathbf{x} + \mathbf{e} \cdot \mathbf{z}$ to find an optimal solution. □

As before we can count and enumerate as in the case of a single objective problem. The arguments are very similar and so details are omitted. The formulation of the set D is based on a construction that improves the one from [23] for encoding Pareto optima which was pointed out by Víctor Blanco (Personal communication, 2007).

4.6 Directions for future work

How do these two algorithms to solve BILP – the one based on parametric integer programming versus the one based on rational generating function techniques – compare in terms of efficiency? It is accurate to say that the algorithm using parametric integer programming is more direct and simpler. Indeed, the rational generating function approach relies on the Projection Lemma which, although it has polynomial running time (in fixed dimension), was until recently thought to be virtually unimplementable in practice. Only recently has the Projection Lemma been implemented in the library `barvinok` developed by Sven Verdoolaege [82]. Moreover, the current implementation appeals to the results in [37], some of the very same results that we use more directly in the development of our first approach (see [53] for more details). Future work would be required to improve implementations of the Projection Lemma to yield a practical solution method for bilevel integer programs. There is also scope in the pure integer setting to compare our algorithms to Moore and Bard [61] and DeNegre and Ralphs [32]. The algorithm for solving BMILPs based on Eisenbrand and Shmonin's approach to parametric integer programming awaits implementation and testing. A first step, of course, would be in implementing Eisenbrand and Shmonin's own algorithm. The authors know of no publicly available implementations.

Chapter 5

A decomposition theorem for bilevel integer programs

5.1 Introduction

In this chapter we consider the following bilevel integer program:

$$\begin{aligned} \inf \quad & f(\mathbf{x}, \mathbf{y}, \mathbf{z}) \\ \text{s.t.} \quad & (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in P \subseteq \mathbb{R}^{n+p+d} \\ & \mathbf{x} \in \operatorname{argmin}\{\, \psi(\mathbf{y}) \cdot \mathbf{x}' : A\mathbf{x}' \leq \pi(\mathbf{z}),\, \mathbf{x}' \in \mathbb{Z}^n \,\} \end{aligned} \tag{5.1}$$

where $A \in \mathbb{Z}^{m \times n}$, f is a real-valued function on P, which is a bounded set, and the functions $\psi : \mathbb{R}^p \to \mathbb{R}^n$ and $\pi : \mathbb{R}^d \to \mathbb{R}^m$ are vector-valued functions. We believe this setting to be an important class of bilevel integer programs, as evidenced in the practical nature of the examples described in Section 5.2.

Intuitively, the challenge of solving (5.1) arises from the fact that the lower level problem is a parametric integer programming problem in both the right-hand side and cost. The challenge can be appreciated by an analogy to parametric linear optimization; that is, problems of the form $\min\{\mathbf{c} \cdot \mathbf{x} : A\mathbf{x} \leq \mathbf{b}\}$ where both $\mathbf{b}$ and $\mathbf{c}$ vary. When only the cost vector $\mathbf{c}$ changes, the polyhedral feasible region remains fixed, giving us some consistent structure which allows us to develop a clear understanding of how the optimal value changes with the

cost vector $\mathbf{c}$. Similarly, when only the right-hand side vector $\mathbf{b}$ changes the dual polyhedron remains fixed and a similar understanding arises. However, when both $\mathbf{b}$ and $\mathbf{c}$ vary a more subtle analysis is required to understand how the optimal value changes (see Section 1 of [79] for more discussion). A full understanding of the situation requires reasoning about when a given column basis of A is both primal and dual feasible as $\mathbf{b}$ and $\mathbf{c}$ change simultaneously. Our problem has an added challenge in that we are concerned with the parametric *integer* programs, where even characterizing the primal and dual feasibility of the column bases of A is not enough to solve the problem. We tackle this conceptual challenge in Section 5.3 below.

This chapter makes the following contributions to the study of bilevel integer programming:

1) The current literature has almost exclusively explored special cases of (5.1) where either the leader's decision affects the follower's unit cost or right-hand side but not both (that is, one of either $\mathbf{y}$ or $\mathbf{z}$ is fixed). To our knowledge, simultaneous changes in the unit cost and right-hand side of the lower level problem have not been studied systematically. The only known algorithms which apply to this more general setting are algorithms for nonlinear discrete bilevel optimization problems. For a summary of such methods see [43]. It should be noted that these methods either provide approximate or heuristic solutions to (5.1), something we improve by demonstrating exact algorithms for certain special cases in this chapter.

2) The main contribution of this chapter, in the authors' view, is our Decomposition Theorem (Theorem 5.3.8) which demonstrates how to decompose (5.1) into a finite number of subproblems involving fixed-unit cost bilevel optimization problems; that is, problems where the follower's objective is a function only of $\mathbf{x}$. Details can be found in Section 5.3, but the following is an informal statement of the result:

Theorem 5.1.1 (**Decomposition Theorem** (informal version))**.** *Consider a bilevel integer*

programming problem of the form (5.1). *There exists a finite number of equivalence classes* $C_i, i = 1, \ldots, N$ *such that the infimum value of* (5.1) *is given by the smallest* ξ_i *arising from the following* N *subproblems:*

$$\begin{aligned} \xi_i := \inf \;\; & f(\mathbf{x}, \mathbf{y}, \mathbf{z}) \\ \text{s.t.} \quad & (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in P \subseteq \mathbb{R}^{n+p+d}, \quad \psi(\mathbf{y}) \in \operatorname{relint} C_i \\ & \mathbf{x} \in \operatorname{argmin}\{\, \mathbf{c}_i \cdot \mathbf{x}' : A\mathbf{x}' \leq \pi(\mathbf{z}),\, \mathbf{x}' \in \mathbb{Z}^n \,\}. \end{aligned}$$

The optimal value of the bilevel integer program, when it exists, is $\min_{i=1,\ldots,N} \xi_i$.

The decomposition is useful not only in deriving new algorithms for solving bilevel integer programs but also analytically for proving existence of optimal solutions, as is demonstrated in our study of two special cases of (5.1) (described below).

3) We use a novel approach to the study of bilevel integer programs, namely, Gröbner basis methods. For an introduction to these methods as used in this paper see [79]. Of particular note is the fact that such methods are well-suited for understanding the parametric nature of the lower-level problem and provide us with a new concept – reduced Gröbner basis – which in some sense correspond to column bases in the linear optimization setting. The theory has both algorithmic and analytical use, which again we demonstrate in our study of two special cases.

The first special case is where all variables are integer, feasible regions are linear and f either affine or nonnegative polynomial. We call this the "pure integer case". In details, we consider the special case of (5.1):

$$\begin{aligned} \min \quad & f(\mathbf{x}, \mathbf{y}, \mathbf{z}) \\ \text{s.t.} \quad & (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in P \subseteq \mathbb{R}^{n+p+d}, \quad \mathbf{y} \in \mathbb{Z}^p, \quad \mathbf{z} \in \mathbb{Z}^d \\ & \mathbf{x} \in \operatorname{argmin}\{\, (C\mathbf{y} + \mathbf{c}) \cdot \mathbf{x}' : A\mathbf{x}' \leq B\mathbf{z} + \mathbf{u},\, \mathbf{x}' \in \mathbb{N}^n \,\}. \end{aligned} \tag{5.2}$$

where P is a polytope, B and C are integer matrices of the appropriate dimensions, and $\mathbf{c}$ and $\mathbf{u}$ are integral vectors of dimension n and m respectively.

This case is the subject of the first subsection of Sections 5.4 and 5.5. We use rational generating functions to derive a polynomial time algorithm (an FPTAS if f is polynomial) for solving each subproblem resulting from the Decomposition Theorem. Although the number of subproblems may be quite large – exponential even in fixed dimension – we discuss some special cases where we are guaranteed exponentially fewer subproblems than in the general case.

The second special case is where we assume some separation in the objectives and constraints of the leader's two decision variables $\mathbf{y}$ and $\mathbf{z}$, which are now assumed to be continuous. This restricted setting is as follows:

$$\begin{aligned} \inf \quad & f(\mathbf{x}, \mathbf{z}) + g(\mathbf{y}) \\ \text{s.t.} \quad & (\mathbf{x}, \mathbf{z}) \in P \subseteq \mathbb{R}^{n+d} \quad \mathbf{y} \in Y \subseteq \mathbb{R}^p \\ & \mathbf{x} \in \operatorname{argmin}\{\, \psi(\mathbf{y}) \cdot \mathbf{x} : A\mathbf{x} \leq \pi(\mathbf{z}),\, \mathbf{x} \in \mathbb{Z}^n \,\} \end{aligned} \tag{5.3}$$

where f is now assumed to be quasiconcave (for reasons discussed in Section 5.4), P a bounded set in $\mathbb{R}^{n+d}$ and Y a bounded set in $\mathbb{R}^p$. We refer to this as the "mixed integer case" since $\mathbf{y}$ and $\mathbf{z}$ are no longer restricted to be integer.

It should be noted from the start that there is one important element of the formulation (5.3) that is more restrictive than other formulations in the literature. In particular, the leader's objective is separable in $\mathbf{y}$ and $\mathbf{z}$ and the upper level constraints $\mathbf{y} \in Y$ and $(\mathbf{x}, \mathbf{z}) \in P$ do not jointly constrain the choice of $\mathbf{y}$ and $\mathbf{z}$. This restriction is central to the results presented below and it is an open question as to whether the approaches here apply when this restriction is relaxed. Despite these restrictions, we believe this model to have useful applicability, such as the model found in Example 5.2.2, which fits this setting.

The main focus of analysis in this latter case is the existence of solutions to problems of this type. Existence is a question that has interested several authors. Vicente et. al. [83] explored the special case of discrete linear bilevel programming problems when $\mathbf{y}$ is fixed and considered the existence of solutions for two cases: one where $\mathbf{z}$ is integer and the other where $\mathbf{z}$ continuous. They found that solutions *always* exist when $\mathbf{z}$ is integer, but *may not* exist when $\mathbf{z}$ is continuous. Our work expands on known existence results by exploring conditions on the functions and constraints in our model (5.3) which guarantee existence of solutions when some upper level variables are continuous. To the authors' knowledge only one other study explores conditions when such solutions exist, which is in the case of a simple knapsack constraint in the lower level [30].

We use Gröbner basis methods to derive new sufficient conditions for existence of optimal solutions for general problems of the form (5.3). These criteria involve assumptions about the functional forms of f, g, ψ and π and the properties of P and Y. Different results arise when the lower level constraints are all equalities and all inequalities. The existence results are contained in Theorem 5.6.3 for equality constrained problems and Theorem 5.6.6 for inequality constrained problems. The Decomposition Theorem plays an important role in both cases.

This chapter is organized as follows. We begin with some motivation in Section 5.2 which describes two examples of bilevel integer programs, one from each of our special cases (5.2) and (5.3). Section 5.3 contains the Decomposition Theorem, and includes some background on polyhedral fans and Gröbner bases. The decomposition result demonstrates the existence of a polyhedral fan which can partition the space of cost vectors $\mathbf{y}$ into a finite number of equivalence classes which correspond to cells.

Up to this point all results are shown for the general case (5.1). From the proof of the Decomposition Theorem it is not clear how to describe and solve each of the subproblems created from the decomposition. At this point we get specific in how we apply the result to

our two special cases. Section 5.4 details how to describe each of the cells in the polyhedral fan and gives an idea of the number of subproblems generated through decomposition. In the special case (5.3) we are able to give a more complete decomposition of the problem, entirely separating the leader's optimization over $\mathbf{y}$ from her choice of $\mathbf{z}$ within each subproblem. Section 5.5 discusses the final computational question associated with the Decomposition Theorem, that of solving each of the subproblems. In the pure integer case we provide a solution method based on the rational generating function method of Barvinok and Woods [8], but in principle, any method for solving bilevel integer programs with fixed unit-cost could be applied. Finally, in Section 5.6 we take up the question of identifying when optimal solutions exist in the mixed integer case. We use the Decomposition Theorem and tackle the question for each subproblem separately. We give checkable sufficient conditions for when optimal solutions are guaranteed to exist.

5.2 Some examples

We begin with a few motivating examples of problems that can be modeled as bilevel as discrete bilevel optimizations of the form (5.1).

Example 5.2.1 (Decentralized shipping (c.f. Section 2.2 of [25]))**.** A shipping company must deliver a shipment of a large stock of items of several types using various vessels. There are t types of items. The weight of each item of type j is w_j and there are n_j items of type j to be shipped. The company has two decentralized divisions: the head office (the leader) and a logistics division (the follower). The head office can secure space on k vessels for the shipment, where vessel ℓ can be outfitted with a maximum weight capacity z_ℓ. The unit cost to the head office of securing capacity z_ℓ on vessel is c_ℓ.

The operating costs of shipping the items is incurred by the logistics division. The cost of shipping one item of type j on vessel ℓ is $p_{j\ell}$. Head office chooses compensation rates

to the logistics division to partially compensate for these costs. The compensation rate for shipping one item of type j on vessel ℓ is $y_{j\ell}$.

The goal of head office is to minimize total compensation costs and capacity costs by deciding on what capacities $\mathbf{z} = (z_1, \ldots, z_k) \in Z \subseteq \mathbb{R}^k$ to secure and what compensation rates $\mathbf{y} = (y_{j\ell}) \in Y$ to offer. The goal of the logistics division is to minimize shipping costs and maximize compensation from head office by choosing the shipment schedule $\mathbf{x} = (x_{j\ell})$ where $x_{j\ell}$ is the number of items of type j to be shipped on vessel ℓ.

We introduce slack variables $x_{q\ell}$ ($q := t+1$) to represent unused weight capacity in vessel ℓ. The formulation is as follows:

$$
\begin{aligned}
\min_{\mathbf{x},\mathbf{y},\mathbf{z}} \quad & \sum_{j=1}^{t}\sum_{\ell=1}^{k} y_{j\ell}x_{j\ell} + \sum_{l=1}^{k} c_\ell z_\ell \\
\text{s.t.} \quad & \mathbf{z} \in Z, \\
& \mathbf{x} \in \operatorname{argmin}\Bigg\{ \sum_{j=1}^{t}\sum_{l=1}^{k}(p_{j\ell} - y_{j\ell})x_{j\ell} : \sum_{l=1}^{k} x_{j\ell} = n_j \quad (j = 1, \ldots, t) \\
& \qquad\qquad \sum_{j=1}^{t} w_j x_{j\ell} + x_{q\ell} = z_k \quad (k = 1, \ldots, n) \quad \mathbf{x} \in \mathbb{N}^{qk} \Bigg\}.
\end{aligned}
\tag{5.4}
$$

This formulation fits our general model (5.1). In fact this problem has a special structure that is discussed in Section 5.4.3. If we let $\mathbf{w} = (w_1, \ldots, w_t, 1)$ denote the row vector of weights, then the constraint matrix of the lower level problem has the form $A^{(k)} = [\mathbf{w}, I]^{(k)}$, a structure we will define in that section as a k-fold matrix (introduced in [25]).

The next example fits the special setting (5.3):

Example 5.2.2. A city planner wants to control traffic on a network by changing capacities $\mathbf{u}$ and tolls $\mathbf{T}$ on arcs. A shipper sends indivisible goods $\mathbf{x}$ on this network and responds to these constraints in order to minimize toll costs $\mathbf{T} \cdot \mathbf{x}$, possibly sending flow along different routes as capacities and tolls change. We assume that the goal of the city planner is to

maximize social welfare: thus, toll revenues ($\mathbf{T} \cdot \mathbf{x}$) are not included in the objective, since toll revenues are a transfer from one party (the shipper) to the other (the city). Instead, the city planner is interested in reducing overall congestion and capacity costs $f(\mathbf{x}, \mathbf{u})$. This is a hierarchical decision-making problem where a *leader* (here the city planner) acts first and influences the decision of a *follower* (here the shipper) and can be modeled as the following bilevel program:

$$\begin{aligned} \min_{\mathbf{x},\mathbf{T}} \quad & f(\mathbf{x}, \mathbf{u}) \\ s.t. \quad & \mathbf{T} \in \Theta, \qquad \mathbf{u} \in U \\ & \mathbf{x} \in \operatorname{argmin}\{\mathbf{T} \cdot \mathbf{x}' : A\mathbf{x}' \leq \boldsymbol{\pi}(\mathbf{u}), \mathbf{x}' \in \mathbb{N}^n\}\} \end{aligned}$$

where Θ is a bounded set of feasible toll values and U is a bounded set of possible capacities and f is a linear or polynomial function. The lower level constraints $A\mathbf{x} \leq \boldsymbol{\pi}(\mathbf{u})$ represent flow-balance and capacity constraints on the network. This fits the setting (5.3), assuming that f is quasiconcave.

5.3 A decomposition theorem

We begin our analysis by considering variable unit cost problems of the general form (5.1) with the only restriction that the lower level problem's constraints are equalities and not inequalities and $\mathbf{x}$ is restricted to be nonnegative. That is, we consider the problem:

$$\begin{aligned} \min \quad & f(\mathbf{x}, \mathbf{y}, \mathbf{z}) \\ \text{s. t.} \quad & (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in P \subseteq \mathbb{R}^{n+p+d} \\ & \mathbf{x} \in \operatorname{argmin}\{\, \psi(\mathbf{y}) \cdot \mathbf{x}' : A\mathbf{x}' = \pi(\mathbf{z}),\ \mathbf{x}' \in \mathbb{N}^n \,\}. \end{aligned} \tag{5.5}$$

The reason for considering equality constrained problems is that the theory used in this chapter has primarily been developed in this setting. We will return to the question of inequality constrained problems at the end of this section.

One notation that we use throughout is to let Z denote to be the projection of P onto its $\mathbf{z}$ coordinates (i.e., $Z = \{\mathbf{z} \in \mathbb{R}^d : (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in P\}$) which defines the overall set of feasible choices of $\mathbf{z}$ for the leader.

Some additional assumptions are required to make the problem well-conditioned for applying Gröbner basis methods, the main tool of analysis:

Assumption 5.3.1. First regarding feasibility, let $\pi(Z)$ denote the range of the function π. Then for (5.5) to be feasible we require $\text{cone}_{\mathbb{N}}(A) \cap \pi(\mathbf{Z})$ to be nonempty where $\text{cone}_{\mathbb{N}}(A)$ is the set of all nonnegative integer linear combinations of the columns of A. Regarding boundedness, we assume A is of full row rank and $ker_{\mathbb{Z}}(A) \cap \mathbb{N}^n = \{0\}$, where $ker_{\mathbb{Z}}(A)$ is the set of integer vectors in the kernel of A. This implies the set of feasible solutions to the lower level problem either empty or is nonempty and bounded for a particular choice of $\mathbf{z}$. This assumption is common in the Gröbner basis literature, see for instance [79].

Section 5.3.1 includes a brief introduction to Gröbner basis methods, focusing on the key concepts and results used to derive equivalence classes for the leader's choice of $\mathbf{y}$.

5.3.1 Background: bases and fans

Many of the concepts and results introduced in this section can be found in [79]. We begin with a few preliminary definitions.

Fix an integer matrix of full row rank $A \in \mathbb{Z}^{m \times n}$. Consider the following instance of an integer program:

$$IP_{A,\mathbf{c}}(\mathbf{b}): \quad \min\{\mathbf{c} \cdot \mathbf{x} : A\mathbf{x} = \mathbf{b},\ \mathbf{x} \in \mathbb{N}^n\}$$

where $\mathbf{c} \in \mathbb{Z}^n$ and $\mathbf{b} \in \mathbb{N}^m$. Let $IP_{A,\mathbf{c}}$ denote the family of integer programs with this fixed A and $\mathbf{c}$ but with varying right-hand side $\mathbf{b}$. Similarly, let IP_A denote the family of integer programs with fixed matrix A and varying $\mathbf{c}$ and $\mathbf{b}$. Note that $IP_{A,\mathbf{c}}(\mathbf{b})$ has a feasible solution if and only if $\mathbf{b} \in \text{cone}_{\mathbb{N}}(A)$.

We call $\mathbf{c}$ *generic* if for each right-hand side $\mathbf{b}$ there is a unique optimum solution to $IP_{A,\mathbf{c}}(\mathbf{b})$, otherwise we say that $\mathbf{c}$ is *non-generic*. For $\mathbf{c}$ generic let $\mathcal{O}_{\mathbf{c}} \subseteq \mathbb{N}^n$ be the set optimal solutions in all programs in the family $IP_{A,\mathbf{c}}$. The complement of this set in $\mathbb{N}^n$ is denoted $\mathcal{N}_{\mathbf{c}}$. It turns out the set $\mathcal{N}_{\mathbf{c}}$ has a simple and useful characterization:

Lemma 5.3.2. *[80] There exists a unique, minimal, set of vectors $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_s \in \mathbb{N}^n$ such that:*

$$\mathcal{N}_{\mathbf{c}} = \bigcup_{k=1}^{s} (\boldsymbol{\alpha}_k + \mathbb{N}^n)$$

The α^k are called the **minimal generators** *of $\mathcal{N}_{\mathbf{c}}$.*

The set $\mathcal{G}_{\mathbf{c}} = \{\boldsymbol{\alpha}_k - \boldsymbol{\omega}_k : k = 1, \ldots s\}$ where $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_s$ are the unique minimal generators of $\mathcal{N}_{\mathbf{c}}$ and $\boldsymbol{\omega}_k = \text{argmin}\{\mathbf{c} \cdot \mathbf{x} : A\mathbf{x} = A\boldsymbol{\alpha}_k\}$ is called the *reduced Gröbner basis* of $IP_{A,\mathbf{c}}$. For each reduced Gröbner basis we associate a closed convex cone

$$\mathcal{K}_{\mathbf{c}} = \{\mathbf{x} \in \mathbb{R}^n : \boldsymbol{\alpha}_k \cdot \mathbf{x} - \boldsymbol{\omega}_k \cdot \mathbf{x} \geq 0\}$$

called a *Gröbner cone*.

We now detail some essential properties of reduced Gröbner bases and Gröbner cones. A *polyhedral fan* in $\mathbb{R}^n$ is a collection of polyhedral cones (called the *cells* of the fan) with the property that if cell C is in the fan then so is every face of C, and if C_1 and C_2 are cells in the fan their intersection is a common face of each. A polyhedral fan in $\mathbb{R}^n$ is *complete* if the union over all the cells in the fan is all of $\mathbb{R}^n$. Given two fans $\mathcal{F}_1$ and $\mathcal{F}_2$ we say $\mathcal{F}_1$ *refines* $\mathcal{F}_2$ if every cell of $\mathcal{F}_1$ is contained in a cell $\mathcal{F}_2$. The *common refinement* of two fans $\mathcal{F}_1$ and

$\mathcal{F}_2$ in $\mathbb{R}^n$, denoted by $\mathcal{F}_1 \cap \mathcal{F}_2$, is the fan of all intersections of cones from $\mathcal{F}_1$ and $\mathcal{F}_2$. See [87] for a thorough introduction to these concepts. The following theorem is a summary of results from [79]:

Theorem 5.3.3. *1. The Gröbner cones are fixed by the matrix A, and as such we call them the* Gröbner cones of A. *Each matrix A admits a* finite *number distinct reduced Gröbner bases and thus there are a* finite *number of Gröbner cones of A.*

2. Gröbner cones are the full-dimensional cells of a polyhedral fan called the Gröbner fan of A. *In particular, the cells of the Gröbner fan are the faces of the Gröbner cones.*

3. Let $P_{\mathbf{b}}^I$ denote the convex hull of the integer points in the polyhedron $P_{\mathbf{b}} = \{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}\}$. The Gröbner fan is the common refinement of the inner normal fans of $P_{\mathbf{b}}^I$ for $\mathbf{b} \in cone_{\mathbb{N}}(A)$. In particular, the Gröbner fan is complete when each $P_{\mathbf{b}}^I$ is a polytope.

For our purposes, the significance of the Gröbner fan A is its relationship to the optimal solutions in the family of integer programs IP_A. The key result is the following generalization of Proposition 3.2 in [79].

Theorem 5.3.4. *[79] Given two cost vectors $\mathbf{c}$ and $\mathbf{c}'$ in $\mathbb{R}^n$ the following are equivalent:*

1. For every $\mathbf{b} \in \text{cone}_{\mathbb{N}}\, A$, the programs $IP_{A,\mathbf{c}}(\mathbf{b})$ and $IP_{\mathbf{c}'}(\mathbf{b})$ have the same set of optimal solutions. That is,

$$\text{argmin}\{\mathbf{c} \cdot \mathbf{x} : A\mathbf{x} = \mathbf{b}, \mathbf{x} \in \mathbb{N}^n\} = \text{argmin}\{\mathbf{c}' \cdot \mathbf{x} : A\mathbf{x} = \mathbf{b}, \mathbf{x} \in \mathbb{N}^n\}$$

2. For every $\mathbf{b} \in \text{cone}_{\mathbb{N}}(A)$, $\text{face}_{\mathbf{c}}(P_{\mathbf{b}}^I) = \text{face}_{\mathbf{c}'}(P_{\mathbf{b}}^I)$, where $\text{face}_{\mathbf{c}}(P_{\mathbf{b}}^I)$ denotes the optimal face of $P_{\mathbf{b}}^I$ in the direction of $\mathbf{c}$.

3. *Both* $\mathbf{c}$ *and* $\mathbf{c}'$ *lie in the relative interior of the same cell of the Gröbner fan of* A*; that is, some face of a Gröbner cone.*

Proof. The equivalence of (5.3.4) and (2) is standard. We complete the proof by showing the equivalence of (2) and (3). The Gröbner fan is the common refinement of the inner normal fans of each polytope P_b^I. Thus two cost vectors lie in the relative interior of the same cell of the Gröbner fan (i.e. a face of a Gröbner cone) if and only if they lie in the relative interior of the same normal cone of each polytope P_b^I. Thus the equivalence of (2) and (3). □

Intuitively, the theorem states that there exists *equivalence classes* for cost vectors $\mathbf{c}$, and these equivalence classes can be represented as the relative interiors of rational polyhedral cones – the faces of Gröbner cones. These cones are rational since each Gröbner cone has facet defining inequalities with integer normal vectors, given by some element $\boldsymbol{\alpha}_k - \boldsymbol{\omega}_k$ of the reduced Gröbner basis. A major contribution from the literature on Gröbner bases is the existence of algorithms to compute these equivalence classes via computation of reduced Gröbner bases and implementations of these algorithms (see for instance the program `CaTS` [47]). We now make note of the obvious fact that any polyhedral fan which *refines* the Gröbner fan yields a "refinement" of equivalence classes for cost vectors $\mathbf{c}$.

Definition 5.3.5 (Consistent)**.** Let A be an $m \times n$ integer matrix. We say a polyhedral fan $\mathcal{F}$ in $\mathbb{R}^n$ is *consistent with respect to* IP_A if for every cell C in $\mathcal{F}$ whenever a pair of vectors $\mathbf{c}$ and $\mathbf{c}'$ both lie in the relative interior of C the integer programs $IP_{A,\mathbf{c}}(\mathbf{b})$ and $IP_{A,\mathbf{c}'}(\mathbf{b})$ have the same set of optimal solutions for every feasible right-hand side $\mathbf{b}$. That is, for every cell C of $\mathcal{F}$, for all $\mathbf{c}, \mathbf{c}' \in \operatorname{relint} C$ and for all $\mathbf{b} \in \operatorname{cone}_{\mathbb{N}}(A)$:

$$\operatorname{argmin}\{\mathbf{c} \cdot \mathbf{x} : A\mathbf{x} = \mathbf{b}, \mathbf{x} \in \mathbb{N}^n\} = \operatorname{argmin}\{\mathbf{c}' \cdot \mathbf{x} : A\mathbf{x} = \mathbf{b}, \mathbf{x} \in \mathbb{N}^n\} \tag{5.6}$$

Theorem 5.3.6. *Let* A *be an* $m \times n$ *integer matrix. A polyhedral fan* $\mathcal{F}$ *is consistent with respect to* IP_A *if and only if it refines the Gröbner fan of* A*.*

Proof. For the only-if direction, suppose $\mathbf{c}$ and $\mathbf{c}'$ lie in the relative interior of the same cell of $\mathcal{F}$. Since $\mathcal{F}$ refines the Gröbner fan of A, this implies $\mathbf{c}$ and $\mathbf{c}'$ lie in the relative interior of a single cell of the Gröbner fan. By Theorem 5.3.4(5.3.4) it follows that $\mathcal{F}$ is consistent.

Conversely, suppose $\mathcal{F}$ is consistent with respect to IP_A. Let C denote a cell of $\mathcal{F}$ and suppose $\mathbf{c}$ and $\mathbf{c}'$ lie in the relative interior of C. Since $\mathcal{F}$ is consistent it follows that $\mathbf{c}$ and $\mathbf{c}'$ satisfy (5.6) for all $\mathbf{b} \in \text{cone}_{\mathbb{N}}(A)$. Thus by Theorem 5.3.4(2), $\mathbf{c}$ and $\mathbf{c}'$ lie in the relative interior of the same cell C' of the Gröbner fan of A. This is true for any two elements of C and thus C is a subset of C'. It then follows that $\mathcal{F}$ is a refinement of the Gröbner fan of A. □

Besides working with reduced Gröbner bases, we also consider another set of integer vectors related to matrix A. The *Graver basis* Gr_A of A is a subset of the integer kernel $ker_{\mathbb{Z}}(A) = \ker(A) \cap \mathbb{Z}^n$ of the matrix A. It was first introduced in [42]. It can be defined as follows: for each orthant O_j of $\mathbb{R}^n$ consider the pointed polyhedral cone $C_j = \ker_{\mathbb{Z}}(A) \cap O_j$. Let H_j be the unique *Hilbert basis* of C_j. The Graver basis is the union of these Hilbert bases over all orthants; that is, $\text{Gr}_A = \cup_j H_j$. The Graver basis is finite and, in general, of size exponential in the input size of the inequality description of A. There exist software implementations which can compute the Graver basis of a matrix (see for instance `4ti2` [1]).

From the Graver basis of A we can define a polyhedral fan which refines the Gröbner fan. To see this, begin by considering the polyhedral fan described by a single hyperplane $H = \{\mathbf{w} \in \mathbb{R}^n : \mathbf{a} \cdot \mathbf{w} = a_o\}$ in $\mathbb{R}^n$ with cells H^+, $H-$ and H, where $H^+ = \{\mathbf{w} \in \mathbb{R}^n : \mathbf{a} \cdot \mathbf{w} \geq a_o\}$ and $H^- = \{\mathbf{w} \in \mathbb{R}^n : \mathbf{a} \cdot \mathbf{w} \leq a_o\}$. The *hyperplane arrangement* $\mathcal{F}_{\mathcal{H}}$ of a finite set of hyperplanes $\mathcal{H}$ in $\mathbb{R}^n$ is the common refinement of all such fans for the hyperplanes in $\mathcal{H}$. In particular, $\mathcal{F}_{\mathcal{H}}$ itself is a polyhedral fan. The *Graver arrangement* of matrix A is the hyperplane arrangement consisting of the hyperplanes in $\mathbb{R}^n$ which are orthogonal to the elements in the Graver basis Gr_A. The key property of Graver arrangements for our purposes is the following:

Proposition 5.3.7 (Proposition 3.13 in [79]). *The Graver arrangement of A is a refinement of the Gröbner fan of A and thus consistent with respect to IP_A.*

Consistent polyhedral fans, and in particular the Gröbner fan and Graver arrangement, will play a major role in the discussion to follow.

5.3.2 The main result

The main result of this section is the identification of a finite number of classes for the cost vectors for the lower level problem and thus "decompose" the problem of simultaneous changes of both right-hand side and cost into a finite number of fixed unit-cost bilevel optimization problems.

To begin we focus our attention on the influence of changes in unit cost and thus define $F(\mathbf{y})$ to be the objective value of (5.5) as a function of the leader's decision variable $\mathbf{y}$. That is,

$$\begin{aligned} F(\mathbf{y}) := \inf \;\; & f(\mathbf{x},\mathbf{y},\mathbf{z}) \\ \text{s.t.} \;\; & (\mathbf{x},\mathbf{y},\mathbf{z}) \in P \subseteq \mathbb{R}^{n+p+d} \\ & \mathbf{x} \in \operatorname{argmin}\{\, \psi(\mathbf{y})\cdot\mathbf{x}' : A\mathbf{x}' \leq \pi(\mathbf{z}),\, \mathbf{x}' \in \mathbb{Z}^n \,\} \end{aligned} \tag{5.7}$$

Note that we have defined $F(\mathbf{y})$ using an infimum rather than a minimum. This is to acknowledge the fact that an infimum value might not be attained.

Let $\mathcal{F}$ denote a polyhedral fan consistent with IP_A and let $C_1,\ldots,C_N$ denote the polyhedral cones comprising the cells of $\mathcal{F}$. For each $i \in \{1,\ldots,N\}$ let

$$\begin{aligned} \xi_i := \inf \;\; & f(\mathbf{x},\mathbf{y},\mathbf{z}) \\ \text{s.t.} \;\; & (\mathbf{x},\mathbf{y},\mathbf{z}) \in P \subseteq \mathbb{R}^{n+p+d}, \quad \psi(\mathbf{y}) \in \operatorname{relint} C_i \\ & \mathbf{x} \in \operatorname{argmin}\{\, \mathbf{c}_i\cdot\mathbf{x}' : A\mathbf{x}' \leq \pi(\mathbf{z}),\, \mathbf{x}' \in \mathbb{Z}^n \,\} \end{aligned} \tag{5.8}$$

where $\mathbf{c}_i$ is an arbitrary fixed unit cost which lies in the relative of the cone C_i. Notice that

to solve for ξ_i we must solve a bilevel optimization with fixed unit cost in the lower level problem. We acknowledge that an optimal solution to this problem might not be attained, one possible reason being that the constraint $\psi(\mathbf{y}) \in \operatorname{relint} C_i$ will involve strict inequalities and thus the feasible region may not be closed.

Now for the main result (that is, a formal statement and proof of Theorem 5.1.1):

Theorem 5.3.8 (**Decomposition Theorem** (formal version))**.** *Consider a bilevel integer programming problem of the form* (5.5)*. Let $C_1, \ldots, C_N$ denote the cells of a polyhedral fan $\mathcal{F}$ in $\mathbb{R}^n$ which is consistent with IP_A. Then*

$$\inf_{\mathbf{y}} F(\mathbf{y}) = \min_{i=1,\ldots,N} \xi_i.$$

Thus the optimal value of the bilevel integer program, when it exists, is $\min_{i=1,\ldots,N} \xi_i$.

Proof. We first claim that $F(\mathbf{y}) = \xi_i$ when $\psi(\mathbf{y}) \in \operatorname{relint} C_i$. When we restrict the choice of $\mathbf{y}$ in this way we clearly satisfy the upper level constraint $\psi(\mathbf{y}) \in \operatorname{relint} C_i$ found in (5.8). Now the fact $\mathcal{F}$ is consistent implies that given any arbitrary cost vector $\mathbf{c}_i \in \operatorname{relint} C_i$

$$\operatorname{argmin}\{\, \psi(\mathbf{y}) \cdot \mathbf{x} : A\mathbf{x} = \pi(\mathbf{z}),\, \mathbf{x} \in \mathbb{Z}^n \,\} = \operatorname{argmin}\{\, \mathbf{c}_i \cdot \mathbf{x} : A\mathbf{x} = \pi(\mathbf{z}),\, \mathbf{x} \in \mathbb{Z}^n \,\}$$

for all $\mathbf{y}$ such that $\psi(\mathbf{y}) \in C_i$. It is then immediate that $F(\mathbf{y}) = \xi_i$. Since $\mathcal{F}$ is a complete (recall our restriction to considering complete fans) $\mathbb{R}^n = \biguplus_{i=1}^n \operatorname{relint} C_i$ (see Chapter 7 of [87]). Thus every value of $\mathbf{y}$ belongs to the relative interior of exactly one cell of $\mathcal{F}$. It then follows that $\inf_{\mathbf{y}} F(\mathbf{y}) = \min_{i=1,\ldots,N} \xi_i$. When the infima are all attained the optimal value is thus $\min_{i=1,\ldots,N} \xi_i$. □

It is also now timely to return to the question of applying our results to the more general inequality constrained problem (5.1). A first idea is to simply apply the common trick to convert a problem with inequality constraints into a problem with only equality constraints

by adding slack variables. An attempt to apply this method here will yield the following difficulty: $\pi(\mathbf{z})$ may take on non-integral values and thus in general we cannot add integer slack variables $\mathbf{s} \in \mathbb{N}^m$ to change the lower level constraints into $A\mathbf{x} + \mathbf{s} = \pi(\mathbf{z})$. In general, we would require $\mathbf{s} \in \mathbb{R}^m$, in which case the lower level problem is a mixed integer program, a case not covered by the theory of this chapter. Hence, a remedy is needed to apply the results from the previous sections to inequality constrained problems.

One solution is to ensure the set of possible right-hand sides $\{\pi(\mathbf{z}) : \mathbf{z} \in Z\} = \pi(Z)$ is a subset of $\mathbb{Z}^m$. One simple way to do this is would be to replace $\pi(\mathbf{z})$ with $\bar{\pi}(\mathbf{z}) = \lfloor \pi(\mathbf{z}) \rfloor$. Notice that since $A\mathbf{x}$ is integer, the constraints $A\mathbf{x} \leq \pi(\mathbf{z})$ and $A\mathbf{x} \leq \bar{\pi}(\mathbf{z})$ are equivalent. This transformation would then allows us to introduce integer slack variables $\mathbf{s} \in \mathbb{N}^m$ so that $A\mathbf{x} + \mathbf{s} = \bar{\pi}(\mathbf{z})$. Thus, we can directly apply the Decomposition Theorem to this problem. Another case when we can ensure it is appropriate to introduce integer slacks is when $\mathbf{z}$ is restricted to be integer and $\pi(\mathbf{z})$ is integer valued for all $\mathbf{z}$ integer. This is a case we will consider in developing algorithms in the following subsection.

It should be noted that adding slack variables increases the dimension of the problem, since now the follower's decision vector $\mathbf{x}' = (\mathbf{x}, \mathbf{s})^\top$. Since some of our results are sensitive to dimension, this can be an important consideration.

5.4 Computing decompositions

In the next two sections we discuss some of the practical implications of the Decomposition Theorem. This section focuses on how to construct the subproblems (5.8) by being more explicit about the constraint $\psi(\mathbf{y}) \in \operatorname{relint} C_i$.

The first step is to compute the cells C_i of a consistent fan. Here we only consider the Gröbner fan or the Graver arrangement, but other methods could be developed for other consistent polyhedral fans.

The next issue step is to describe in a meaningful way the equivalence classes for $\mathbf{y}$; that is, the relative interiors of the C_i. By "meaningful", we mean in a way that provides some hope for deciding the existence of optimal solutions to (5.8), and for finding optimal solutions if they exist. This is a difficult issue, and we tackle it in two special subcases. The first is a setting where all the variables are integer, and second is a mixed integer setting. They are explored in detail below.

To close the section we address the question of just how many subproblems may arise by asking how the number of cells in a consistent fan grows with the input size. The answer is discouraging, namely that the number of cells can grow exponentially in the input size, even when the dimension is fixed. We close the section with a special case where the number of equivalence classes may be exponentially smaller than the general case. The issue of how to actually compute ξ_i once we have given an explicit description of relint C_i is the subject of the next section.

5.4.1 Describing cells

We take up the first issue in detail for constructing the Graver arrangement of A. The key result is a general algorithm for computing hyperplane arrangements of a set of vectors.

Theorem 5.4.1 (Theorem 3.3, [36]). *Consider a finite set of M hyperplanes $\mathcal{H}$ in $\mathbb{R}^n$ specified by their normal vectors. Then there exists an algorithm that constructs the following combinatorial representation of the hyperplane arrangement $\mathcal{F}_{\mathcal{H}}$ of $\mathcal{H}$:*

(i) an incidence lattice of the cells in $\mathcal{F}_{\mathcal{H}}$ and

(ii) for each cell C in $\mathcal{F}_{\mathcal{H}}$:

(a) a vector in the interior of C,

(b) a list of supporting hyperplanes of C given by a subset of vectors in $\mathcal{H}$.

This algorithm runs in $O(M^n)$ time.

We can apply Theorem 5.4.1 directly to the set of integer vectors in the Graver basis of A to represent the cells of the Graver arrangement. The question of how to construct the Gröbner fan has been explored in detail in [45] and implemented in [47], which uses facts from toric algebra that will not be explored in detail here.

5.4.2 Describing equivalence classes

Pure integer case

In the pure integer case subproblems are of the form:

$$\begin{aligned} \xi_i := \min \quad & f(\mathbf{x}, \mathbf{y}, \mathbf{z}) \\ \text{s.t.} \quad & (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in P \subseteq \mathbb{R}^{n+p+d}, \quad (C\mathbf{y} + \mathbf{c}) \in \operatorname{relint} C_i, \quad \mathbf{y} \in \mathbb{Z}^p, \quad \mathbf{z} \in \mathbb{Z}^d \\ & \mathbf{x} \in \operatorname{argmin}\{\, \mathbf{c}_i \cdot \mathbf{x}' : A\mathbf{x}' \leq B\mathbf{z} + \mathbf{u},\, \mathbf{x}' \in \mathbb{N}^n \,\} \end{aligned} \tag{5.9}$$

To solve this problem we will use a generating approach similar to that found in Chapter 4. The goal here is to convert the constraint $(C\mathbf{y} + \mathbf{c}) \in \operatorname{relint} C_i$ into something amenable to generating functions; that is, a closed polyhedral description. Via Theorem 5.4.1 each cell C_i is given by a set of supporting hyperplanes, which in the case of both the Graver arrangements and the Gröbner fan have integral supporting vectors. Thus, C_i is a rational polyhedral cone expressible as $C_i = \{\mathbf{w} \in \mathbb{R}^n : M_i\mathbf{w} \geq 0\}$ where M_i is an integer matrix and $\operatorname{relint} C_i = \{\mathbf{w} \in \mathbb{R}^n : M_i\mathbf{w} > 0\}$. Since $\psi(\mathbf{y}) = C\mathbf{y} + \mathbf{c}$ is an integral vector inside the relative interior of C_i it follows that

$$\psi(\mathbf{y}) \in \operatorname{relint} C_i \cap \mathbb{Z}^n = \{\mathbf{w} \in \mathbb{R}^n : M_i\mathbf{w} \geq 1\}.$$

Hence the constraint $(C\mathbf{y} + \mathbf{c}) \in \operatorname{relint} C_i$ can be translated directly into a constraint on the leader's choice of $\mathbf{y}$; namely that $\mathbf{y}$ lie in the set

$$Y_i = Y \cap \{\mathbf{w} \in \mathbb{R}^n : M_i C\mathbf{y} \geq -M_i\mathbf{c} + 1\}. \tag{5.10}$$

Note that Y_i is a closed polyhedron, and thus we now have a closed description of the feasible region of (5.9). See for Section 5.5 for a method to compute ξ_i given this description of Y_i.

Mixed integer case

Now consider a special setting where $\mathbf{y}$ and $\mathbf{z}$ are continuous. The main idea here is use the continuity of $\mathbf{y}$ to avoid consideration of lower dimensional equivalence classes and overcome the difficulties of having open equivalence classes. We analyze the problem with equality lower level constraints. We may assume equality constraints without loss of generality by similar reasoning used in the previous section.

$$\begin{aligned} \min \quad & f(\mathbf{x}, \mathbf{z}) + g(\mathbf{y}) \\ \text{s. t.} \quad & (\mathbf{x}, \mathbf{z}) \in P \subseteq \mathbb{R}^{n+d}, \quad \mathbf{y} \in Y \subseteq \mathbb{R}^p \\ & \mathbf{x} \in \operatorname{argmin}\{\, \psi(\mathbf{y}) \cdot \mathbf{x} : A\mathbf{x} = \pi(\mathbf{z}),\, \mathbf{x} \in \mathbb{N}^n \,\} \end{aligned} \tag{5.11}$$

where $A \in \mathbb{Z}^{m \times n}$, the functions f and g are real-valued functions on P and Y respectively. The sets P and Y are bounded and the functions $\psi : \mathbb{R}^p \to \mathbb{R}^n$ and $\pi : \mathbb{R}^d \to \mathbb{R}^n$ are vector-valued real functions. As before we let Z denote the projection of P onto its $\mathbf{z}$ coordinates (i.e., $Z = \{\mathbf{z} \in \mathbb{R}^d : (\mathbf{x}, \mathbf{z}) \in P\}$). Lastly, when working with (5.11) we assume that f is quasiconcave. The reason for this assumption is apparent in the proofs that follow.

We show that the separation of $\mathbf{y}$ and $\mathbf{z}$ in the leader's objective translates to a complete separation from the perspective of the leader for optimizing her choices. Note, that in (5.11) the only connection between $\mathbf{y}$ and $\mathbf{z}$ is implicit through the follower's objective function.

Via a modification of the Decomposition Theorem (discussed below) this implicit connection can be severed. An alternate argument is developed to achieve this result, wherein we also show how we need only consider full dimensional equivalence classes. The details of this section refer specifically to equivalence classes deriving from the Gröbner fan. A similar argument would follow for any consistent fan.

Let $\mathcal{K}_1, \ldots \mathcal{K}_t$ denote the full-dimensional Gröbner cones (or simply the Gröbner cones) of A. These are finite in number by Theorem 5.3.3. In addition, let $\mathbf{x}_i(\mathbf{b})$ denote the unique optimal solution to $IP_{\mathbf{c}}(\mathbf{b})$ when $\mathbf{c}$ is a generic cost vector in the interior of $\mathcal{K}_i$. That is, let

$$\mathbf{x}_i(\mathbf{b}) = \text{argmin}\{\mathbf{c} \cdot \mathbf{x} : A\mathbf{x} = \mathbf{b}, \mathbf{x} \in \mathbb{N}^n\} \quad \text{when } c \in \text{relint}\, C_i \tag{5.12}$$

We denote the set of Gröbner cones which contain $\mathbf{c}$ by $I(\mathbf{c}) = \{i : \mathbf{c} \in \mathcal{K}_i\} \subseteq \{1, \ldots, t\}$. This set is a singleton when $\mathbf{c}$ is generic. Our approach to handling the case of non-generic cost vectors is to employ the following lemma which applies to both generic and non-generic cost vectors:

Lemma 5.4.2 (**Convex Hull Lemma**). *Given a cost vector* $\mathbf{c}$ *and right-hand side* $\mathbf{b} \in \text{cone}_{\mathbb{N}}(A)$ *it follows that the set of optimal solutions to* $IP_{\mathbf{c}}(\mathbf{b})$ *has the following description:*

$$\text{argmin}\{\mathbf{c} \cdot \mathbf{x} : A\mathbf{x} = \mathbf{b}, \mathbf{x} \in \mathbb{N}^n\} = \text{conv}\{\mathbf{x}_i(\mathbf{b}) : i \in I(\mathbf{c})\} \cap \mathbb{N}^n.$$

In particular, if two cost vectors $\mathbf{c}$ *and* $\mathbf{c}'$ *have* $I(\mathbf{c}) = I(\mathbf{c}')$ *then* $IP_{\mathbf{c}}(\mathbf{b})$ *and* $IP'_{\mathbf{c}}(\mathbf{b})$ *have the same set of optimal solutions for each* $\mathbf{b} \in \text{cone}_N(A)$.

Proof. Recall the notation that $P^I_{\mathbf{b}}$ denotes the convex hull of the integer points in the polytope $\{\mathbf{x} \in \mathbb{R}^n : A\mathbf{x} = \mathbf{b}\}$. Then $\text{argmin}\{\mathbf{c} \cdot \mathbf{x} : A\mathbf{x} = \mathbf{b}, \mathbf{x} \in \mathbb{N}^n\}$ is the set of integer points in the optimal face of P^I_b in the direction of c. Since the Gröbner fan is a refinement of the inner normal fan of $P^I_{\mathbf{b}}$, the set $\{\mathbf{x}_i(\mathbf{b}) : i \in I(\mathbf{c})\}$ contains the vertices of this face.

The result then follows. □

For this subsection redefine the function $F(\mathbf{y})$ as follows:

$$\begin{aligned} F(\mathbf{y}) = \inf_{\mathbf{x},\mathbf{z}} \quad & f(\mathbf{x},\mathbf{z}) + g(\mathbf{y}) \\ s.t. \quad & (\mathbf{x},\mathbf{z}) \in P \subseteq \mathbb{R}^{n+d} \\ & \mathbf{x} \in \operatorname{argmin}\{\psi(\mathbf{y}) \cdot x : A\mathbf{x} = \pi(\mathbf{z}),\ \mathbf{x} \in \mathbb{N}^n\} \end{aligned} \tag{5.13}$$

We make use of the Convex Hull Lemma to yield the following:

Lemma 5.4.3. *For all* $\mathbf{y} \in Y$

$$F(\mathbf{y}) = \min\{\xi_i : i \in I(\psi(\mathbf{y}))\} + g(\mathbf{y}).$$

where

$$\begin{aligned} \xi_i := \inf \quad & f(\mathbf{x},\mathbf{z}) \\ \text{s. t.} \quad & (\mathbf{x},\mathbf{z}) \in P, \psi(\mathbf{y}) \in \mathcal{K}_i \\ & \mathbf{x} \in \operatorname{argmin}\{\, \mathbf{c}_i \cdot \mathbf{x}' : A\mathbf{x}' = \pi(\mathbf{z}),\, \mathbf{x}' \in \mathbb{N}^n \,\} \end{aligned}$$

with $\mathbf{c}_i$ *a generic cost vector in Gröbner cone* $\mathcal{K}_i$*.*

Proof. We begin by expressing $F(\mathbf{y}) = \theta(\mathbf{y}) + g(\mathbf{y})$ where

$$\theta(\mathbf{y}) := \inf_{\mathbf{x},\mathbf{z}} \left\{ f(\mathbf{x},\mathbf{z}) : \mathbf{z} \in Z, \mathbf{x} \in \operatorname{argmin}_{\tilde{\mathbf{x}}}\{\psi(\mathbf{y}) \cdot \tilde{\mathbf{x}} : A\tilde{\mathbf{x}} = \pi(\mathbf{z}), \tilde{\mathbf{x}} \in \mathbb{N}^n\} \right\}$$

This function can be manipulated as follows:

$$\begin{aligned}
\theta(\mathbf{y}) &= \inf_{\mathbf{x},\mathbf{z}} \{f(\mathbf{x},\mathbf{z}) : \mathbf{x} \in \operatorname{argmin}_{\tilde{\mathbf{x}}}\{\psi(\mathbf{y})\cdot\mathbf{x} : A\mathbf{x} = \pi(\mathbf{z})\}, \mathbf{z} \in Z\} && (5.14)\\
&= \inf_{z\in Z}\left\{\inf_{x}\left\{f(\mathbf{x},\mathbf{z}) : \mathbf{x} \in \operatorname{conv}\{\mathbf{x}_i(\mathbf{z}) : i \in I(\psi(\mathbf{y}))\cap\mathbb{N}^n\}\right\}\right\} && (5.15)\\
&= \inf_{z\in Z}\left\{\min_{i\in I(\psi(\mathbf{y}))} f(\mathbf{x}_i(\mathbf{z}),\mathbf{z})\right\} && (5.16)
\end{aligned}$$

From (5.14) to (5.15) we apply the Convex Hull Lemma. From (5.15) to (5.16) we use the fact that f is a quasiconcave function and so its minimum occurs at the extreme points of its feasible region. We now change the order of optimization over i and $\mathbf{z}$ and reintroduce the optimization in $\mathbf{x}$. Let $\mathbf{c}_i$ be any cost vector in the interior of $\mathcal{K}_i$, then continuing from (5.16):

$$\begin{aligned}
\theta(\mathbf{y}) &= \min_{i\in I(\psi(\mathbf{y}))} \inf_{\mathbf{x},\mathbf{z}}\{f(\mathbf{x},\mathbf{z}) : \mathbf{z} \in Z, \mathbf{x} \in \operatorname{argmin}_{\mathbf{x}\in\mathbb{N}^n}\{\mathbf{c}_i\cdot\mathbf{x} : A\mathbf{x} = \pi(\mathbf{z})\}\} && (5.17)\\
&= \min_{i\in I(\psi(\mathbf{y}))} \xi_i && (5.18)
\end{aligned}$$

From (5.16) to (5.17) we expand the definition of $\mathbf{x}_i(\mathbf{z})$ and combine the optimization in $\mathbf{x}$ and $\mathbf{z}$. From (5.17) to (5.18) we note the definition of ξ_i. The result follows. □

In contrast to the analysis in Section 5.3, we need only consider full dimensional Gröbner cones, which follows from the Convex Hull Lemma and the fact that we assume f is quasiconcave.

This analysis yields the following alternate decomposition result:

Theorem 5.4.4. *A bilevel integer program of the form* (5.11) *satisfying Assumption 5.3.1 can be decomposed into a finite number of paired subproblems, one pair of subproblems for each Gröbner cone $\mathcal{K}_i$. Each pair includes:*

1. *A fixed-cost bilevel optimization problem to find the optimal choice of* $\mathbf{z}$ *and* $\mathbf{x}$*:*

$$\begin{aligned} &\inf \ f(\mathbf{x}, \mathbf{z}) \\ \text{s.t.} \quad &(\mathbf{x}, \mathbf{z}) \in P \subseteq \mathbb{R}^{n+d} \\ &\mathbf{x} \in \operatorname{argmin}\{\, \mathbf{c}_i \cdot \mathbf{x}' : A\mathbf{x}' \leq \pi(\mathbf{z}),\, \mathbf{x}' \in \mathbb{Z}^n \,\} \end{aligned}$$

2. *An optimization problem for* y *to optimize* $g(\mathbf{y})$ *when* $\mathbf{y}$ *is constrained so that* $\psi(\mathbf{y})$ *lies in* $\mathcal{K}_i$*; that is,*

$$\begin{aligned} &\inf \ g(y) \\ \text{s.t.} \quad &y \in Y(i) \end{aligned}$$

where

$$\begin{aligned} Y(i) &= \{\mathbf{y} \in Y : i \in I(\psi(\mathbf{y}))\} \\ &= \{\mathbf{y} \in Y : \psi(\mathbf{y}) \in \mathcal{K}_i\}. \end{aligned}$$

These optimization problems are called the "cone optimization" subproblems.

Proof. Using the notation of the previous results it suffices to show:

$$\inf_{\mathbf{y} \in Y} F(\mathbf{y}) = \min_{i \in I} \{\xi_i + \inf_{\mathbf{y} \in Y(i)} g(\mathbf{y})\} \tag{5.19}$$

where $Y(i) = \{\mathbf{y} \in Y : \psi(\mathbf{y}) \in \mathcal{K}_i\}$. This in turn follows from the previous proposition and a simple exchange of minimizations:

$$\begin{aligned} \inf_{\mathbf{y} \in Y} F(\mathbf{y}) &= \inf_{\mathbf{y} \in Y} \Big\{ \min_{i \in I(\psi(\mathbf{y}))} \xi_i + g(\mathbf{y}) \Big\} \\ &= \min_{i \in I} \left\{ \inf_{\mathbf{y} \in Y(i)} \big\{ \xi_i + g(\mathbf{y}) \big\} \right\} \\ &= \min_{i \in I} \big\{ \xi_i + \inf_{y \in Y(i)} g(\mathbf{y}) \big\} \end{aligned} \tag{5.20}$$

□

The significance of this result is that we can separate the leader's choice of $\mathbf{y}$ from her choice of $\mathbf{z}$ in the sense that they can be optimized independently in these two subproblems, provided we isolate the choice of $\mathbf{y}$ so that $\psi(\mathbf{y})$ stays within some Gröbner cone K_i. Note that the problems now do not involve reasoning about the relative interiors of cones, but instead the full dimensional Gröbner cones $\mathcal{K}_i$.

5.4.3 The number of subproblems

As discussed above, the decomposition is sensitive to the *number* of cells, since each cell gives rise to a fixed-unit cost subproblem. An example from Jensen [48] (Example 1.0.2) shows that even with $n = 3$ there can be an exponential number of cells in the Gröbner fan of A, and thus at least this many for any refinement. Thus, we are motivated to search for cases where the number of cells in the arrangement is relatively small, leading ideally to a more computationally tractable number of fixed unit-cost subproblems. We now describe a special case where there is fewer, albeit still exponentially many, subproblems.

The special case is when the lower level constraint matrix arises with the following special structure:

Definition 5.4.5 (k-fold bilevel integer program)**.** Fix any pair of integer matrices R and S with the same number of columns, of dimensions $r \times q$ and $s \times q$ respectively. The *k-fold*

matrix of the ordered pair (R, S) is the following $(s + qr) \times kq$ matrix,

$$[R, S]^{(k)} = \begin{pmatrix} R & R & R & \cdots & R \\ S & 0 & 0 & \cdots & 0 \\ 0 & S & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & S \end{pmatrix}$$

A *k-fold bilevel integer program* is a bilevel integer program of the form (5.5) where the lower level problem can be expressed as:

$$\min\{\psi(\mathbf{y}) \cdot \mathbf{x} : A\mathbf{x} = \pi(\mathbf{z}),\ \mathbf{x} \in \mathbb{N}^n\} \tag{5.21}$$

where $A = [R, S]^{(k)}$ is a k-fold matrix for fixed matrices R and S.

These bilevel integer programs put into a bilevel setting a special class of integer programs introduced by De Loera et. al. [25] called *k-fold integer programs*. Besides having attractive computational properties, k-fold bilevel integer programs can be used to model a wide range of applications, such as Example 5.2.1. An important characteristic of k-fold bilevel integer programs is the following:

Theorem 5.4.6 (Theorem 4.2 in [25])**.** *Fix a pair of integer matrices R and S of dimensions $r \times q$ and $s \times q$ respectively. Then there is a polynomial time algorithm that, given k computes the Graver basis Gr_A of the k-fold matrix $A = [R, S]^{(k)}$. In particular, the cardinality and the bit size of Gr_A are bounded by a polynomial in k.*

In general, the number of Graver basis elements M of an arbitrary matrix is exponential in size, and thus according to Theorem 5.4.1 we expect a doubly exponential number of cells, $O(M^n)$. However, in the k-fold setting our Graver bases are manageable in size even when

the dimension varies and as a result the number of cells is exponential only in the dimension. This represents a potential for exponentially fewer subproblems as compared to the general case.

5.5 Solving the subproblems

We now turn to the next computational hurdle, that of actually solving each subproblem for ξ_i in (5.8). We discuss the issue in our two special settings.

5.5.1 Pure integer setting

Letting $P_i = \{(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in \mathbb{R}^{n+p+d} : (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in P, \mathbf{y} \in Y_i\}$ where Y_i is defined as in (5.10), we can restate (5.2) as:

$$\begin{aligned} \xi_i := \min \quad & f(\mathbf{x}, \mathbf{y}, \mathbf{z}) \\ \text{s.t.} \quad & (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in P_i \subseteq \mathbb{R}^{n+p+d}, \quad \mathbf{y} \in \mathbb{Z}^p, \quad \mathbf{z} \in \mathbb{Z}^d \\ & \mathbf{x} \in \operatorname{argmin}\{\, \mathbf{c}_i \cdot \mathbf{x}' : A\mathbf{x}' \leq B\mathbf{z} + \mathbf{u},\, \mathbf{x}' \in \mathbb{N}^n \,\} \end{aligned} \tag{5.22}$$

The upper level constraint can be seen to be polytopal, since P_i is a polytope. We may now apply any method given in the literature that solves bilevel integer programs, for instance [32, 61]. We adapt here the method of Chapter 4. Let $S_i = P_i \cap \mathbb{Z}^{n+p+d}$ and denote by $\mathcal{B}$ the set of bilevel integer feasible solutions of (5.22); that is,

$$\mathcal{B} = \{(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in S_i : \mathbf{x} \in \operatorname{argmin}\{\, \mathbf{c}_i \cdot \mathbf{x}' : A\mathbf{x}' \leq B\mathbf{z} + \mathbf{u},\, \mathbf{x} \in \mathbb{Z}^n \,\}\} \tag{5.23}$$

Theorem 5.5.1. *The set $\mathcal{B}$ has a short rational generating function encoding, and it can be found in polynomial time when the dimension $n + p + d$ is fixed.*

Proof. Let $\Omega = \{(\mathbf{x}, \mathbf{y}, \mathbf{z}) \in S_i : A\mathbf{x} \leq \mathbf{B}\mathbf{z} + u\}$ and

$$D = \text{proj}_{(\mathbf{x},\mathbf{y},\mathbf{z})} \left\{ (\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{x}') : \begin{array}{r} (\mathbf{x}, \mathbf{y}, \mathbf{z}) \in S_i,\ (\mathbf{x}', \mathbf{y}, \mathbf{z}) \in S_i, \\ A\mathbf{x} = \mathbf{B}\mathbf{z} + \mathbf{u},\ A\mathbf{x}' = B\mathbf{z} + \mathbf{u}, \\ \mathbf{c}_i \cdot \mathbf{x}' \leq \mathbf{c}_i \cdot \mathbf{x} - 1 \end{array} \right\}$$

The set of bilevel integer feasible solutions is $\mathcal{B} = \Omega \setminus D$ can be encoded as a short rational generating function by similar reasoning found in the proof of Theorem 2.3.8. □

It is now a simple task to work with the generating function encoding of $\mathcal{B}$ in order to find ξ_i. The following is a summary of some key results.

Corollary 5.5.2. *Consider a bilevel integer program of the form* (5.2) *and with subproblems of the form* (5.22) *for* $i = 1, \ldots, N$. *Then*

(i) There exists a polynomial time algorithm to find ξ_i *for each* $i \in \{1, \ldots, N\}$ *when leader's objective* f *is linear and the dimension* $n + p + d$ *is fixed.*

(ii) There exists a polynomial time algorithm to find an optimal solution *to each subproblem when the leader's objective* f *is linear.*

(iii) There exists an FPTAS to calculate ξ_i *for each* $i = 1, \ldots, N$ *when the leader's objective* f *is a non-negative polynomial on* P, *and the dimension* $n + p + d$ *is fixed.*

Thus, overall, although there may be exponentially many subproblems in this case, each one is solvable in polynomial time. This result could potentially be used as the basis of an branch-and-bound method to solve the overall problem (5.2), where we can prune certain subproblems using upper and lower bounds similar to that proposed in Section 3.6.2. This is an area for future explorations.

5.5.2 Mixed integer case

In this case, we get pairs of subproblems, those only involving $\mathbf{x}$ and $\mathbf{z}$ in a fixed-unit cost bilevel integer programs and those involving $\mathbf{y}$ is a general optimization problem. Any method used to solve such bilevel integer programs can be applied here provided the other special conditions of the method of met. Depending on the properties of Y and g, different optimization methods could be used to solve the cone optimization subproblems.

As mentioned in the introduction, the primary focus for this case is not algorithms to solve the problem so much as the structure and existence of optimal solutions. This is the topic we explore next.

5.6 Some existence results

This section explores conditions on our input data – f,g, ψ, π, P and Y – whereby optimal solutions to (5.3) are guaranteed to exist. Existence is a separate issue from the decomposition, which holds even when infima are not obtained. Throughout this section we assume that $\mathbf{x}$ lies in some overall bounded discrete set $X \in \mathbb{Z}^d$ which does not depend on the leader's choice of $\mathbf{z}$.

5.6.1 Equality constrained case

We begin by first exploring the existence of solutions to the equality constrained problem (5.11) which will be a base for the more general case (5.3).

The approach is to first show existence of solutions to the fixed-cost bilevel subproblems. Fix a cone $\mathcal{K}_i$ and let $\mathbf{c}_i$ be fixed and lie in the interior of $\mathcal{K}_i$. Recall that the ith fixed-cost

bilevel subproblem has the form:

$$\begin{aligned} \inf \quad & f(\mathbf{x}, \mathbf{z}) \\ \text{s.t.} \quad & (\mathbf{x}, \mathbf{z}) \in P, \\ & \mathbf{x} \in \operatorname{argmin}\{\, \mathbf{c}_i \cdot \mathbf{x}' : A\mathbf{x}' = \pi(\mathbf{z}),\, \mathbf{x}' \in X \,\} \end{aligned} \tag{5.24}$$

We assume without loss of generality throughout this subsection that $\mathbf{x}$ is nonnegative. Recall we can take $\mathbf{c}_i$ to be generic, and thus each value of $\pi(\mathbf{z}) \in \operatorname{cone}_{\mathbb{N}}(A)$ defines a unique optimal solution to the lower level problem: $\min\{\mathbf{c}_i \cdot \mathbf{x} : A\mathbf{x} = \pi(\mathbf{z}), \mathbf{x} \in X, \mathbf{x} \geq \mathbf{0}\}$. As before we denote this unique optimal solution by $\mathbf{x}_i(\pi(\mathbf{z}))$. Let $\mathcal{O}_{\mathbf{c}_i}$ denotes the set of optimal solutions as the right hand side $\pi(\mathbf{z})$ varies. For simplicity denote this set by $\mathcal{O}_i$.

Observe $\mathcal{O}_i \cap X$ contains all of those $\mathbf{x} \in X$ that are optimal for some right-hand side $\pi(\mathbf{z})$. Using this notation, problem (5.24) can be restated as:

$$\begin{aligned} \xi_i := \inf \quad & f(\mathbf{x}, \mathbf{z}) \\ \text{s.t.} \quad & (\mathbf{x}, \mathbf{z}) \in P, \\ & \mathbf{x} \in \mathcal{O}_i \cap X, \\ & A\mathbf{x} = \pi(\mathbf{z}) \end{aligned} \tag{5.25}$$

Lemma 5.6.1. *Optimal solutions to the ith bilevel subproblem* (5.24) *exist if P is compact and π is a continuous function.*

Proof. We have already assumed that f is quasiconcave, and thus lower semicontinuous and so showing the compactness of the feasible region suffices for showing existence of optimal solutions. We see that $\mathcal{O}_i \cap X$ is a bounded discrete set. If π is continuous then its level sets are closed and hence the set $\{(\mathbf{x}, \mathbf{z}) \in X \times \mathbb{R}^d : A\mathbf{x} = \pi(\mathbf{z})\}$ is closed. Lastly, since P is assumed to be closed, the feasible region to (5.25) is a closed set and the result then follows. □

We now turn to the question of existence for the cone optimization subproblems.

Lemma 5.6.2. *Fix a Gröbner cone $\mathcal{K}_i$ and consider the optimization problem* $\inf_{\mathbf{y}\in Y(i)} g(\mathbf{y})$. *Optimal solutions to this problem exist if Y is compact, ψ is continuous and g is lower semicontinuous.*

Proof. We first determine the properties of the feasible set $Y(i)$.

If Y is compact and ψ is continuous, the set $Y(i)$ is closed. Thus, $Y(i)$ is a closed subset of Y and therefore compact. It follows that the cone optimization subproblem attains its infimum when g is lower semicontinuous. □

We are now ready to provide conditions under which the bilevel problem (5.11) has optimal solutions. Simply combining the results of the last two lemmas yields the following:

Theorem 5.6.3 (**Existence Theorem**)**.** *An optimal solution to* (5.11) *exists if both Y and Z are compact, ψ and π are both continuous and g is lower semicontinuous. Recall, throughout we have assumed that f is quasiconcave.*

5.6.2 Inequality constrained case

Now we attempt to apply the Existence Theorem to the more general setting where inequality constraints are permitted in the lower level problem; i.e., bilevel integer programs of the form (5.3).

Recall at the end of Section 5.3 our initial approach to handle inequalities was to replace $\pi(\mathbf{z})$ with $\bar{\pi}(\mathbf{z}) = \lfloor \pi(\mathbf{z}) \rfloor$ and this allowed us to introduce integer slack variables $\mathbf{s} \in \mathbb{N}^m$ so that $A\mathbf{x} + \mathbf{s} = \bar{\pi}(\mathbf{z})$. However, in this case the following difficulty arises when applying the Existence Theorem (Theorem 5.6.3) after introducing slack variables: even if π is a continuous function, $\lfloor \pi(\mathbf{z}) \rfloor$ may not be. More directly we see that we can only introduce integer slack variables if the set of possible right-hand sides $\pi(Z)$ is a subset of $\mathbb{Z}^m$ in which

case π is not a continuous function and thus violating a key condition of the Existence Theorem. An alternate approach must be taken establish the existence of optimal solutions. The following example demonstrates the need for stronger conditions in the inequality case.

Example 5.6.4. Consider the following instance of (5.3):

$$f(x,z) = -2x + z, \qquad g(y) = 0, \qquad h(z) = z, \qquad Y = \emptyset \qquad Z = [0,4]$$

$$A = \begin{pmatrix} -3 \\ -1 \end{pmatrix}, \qquad \pi(\mathbf{z}) = \begin{pmatrix} 0 \\ -1 \end{pmatrix} + \begin{pmatrix} -1 \\ 0 \end{pmatrix} z, \qquad \psi(\mathbf{y}) = 1$$

With some calculation we find that:

$$\text{argmin}_{x \in \mathbb{N}^n} \left\{ x : Ax \leq \pi(\mathbf{z}) \right\} = \begin{cases} 1, & 0 \leq z \leq 3 \\ 2, & 3 < z \leq 4 \end{cases}$$

In order to minimize her objective the leader's goal is to the choose z of minimal cost so that the follower chooses $x = 2$. This is where the complication arises since

$$\inf_{x,z \in [0,4]} \{-2x + z : x \in \text{argmin}\{x : Ax \leq \pi(z)\}\}$$

has an infimum value of -1, but this infimum cannot be obtained.

This example demonstrates that even if the functions f, g, h, ψ and π are all continuous and the sets Y and Z are both compact an optimal solution may not exist.

More refined assumptions on the structure of these functions and sets are required to ensure existence of optimal solutions. As in the equality case we assume there is an overall bounding set X for the choice of $\mathbf{x}$, whereas now we no longer assume that $\mathbf{x} \geq 0$. The following is a list of sufficient conditions for the existence of solutions to (5.3):

(A1) π is upper semicontinuous and P is compact.

(A2) The set of points in $\text{cone}_{\mathbb{N}}(A)$ that lead to feasible solutions

$$\pi^{\leq} = \{A\mathbf{x} : \mathbf{x} \in X, A\mathbf{x} \leq \pi(\mathbf{z}) \textit{ for some } \mathbf{z} \textit{ where } (\mathbf{x}, \mathbf{z}) \in P\}$$

is a subset of $\pi(\mathbf{Z})$. That is, for every $\mathbf{x} \in X$ such that $A\mathbf{x} \leq \pi(\mathbf{z})$ for some $\mathbf{z} \in Z$ there exists a $\tilde{\mathbf{z}} \in Z$ such that $A\mathbf{x} = \pi(\tilde{\mathbf{z}})$.

(A3) (Comonotonicity) For all $\mathbf{z}, \mathbf{z}' \in Z$ and $\mathbf{x} \in X$:

$$\pi(\mathbf{z}) \leq \pi(\mathbf{z}') \implies f(\mathbf{x}, \mathbf{z}) \leq f(\mathbf{x}, \mathbf{z}')$$

.

(A4) For all $\mathbf{y} \in Y$, $\psi(\mathbf{y})$ is in the cone generated by the columns of $-A$; that is, there exists a vector $\mathbf{u_y} \geq 0$ in $\mathbb{R}^m$ such that $\psi(\mathbf{y}) = -\mathbf{u_y}A$.

(A5) g is lower semicontinuous and Y is compact.

Later in this section we will discuss when these assumptions might be reasonable. For now we focus on establishing the existence of optimal solutions to the inequality constrained problem under these assumptions.

One difficulty when thinking about the inequality case is that the theory in Section 5.3.1 has been developed exclusively in the equality setting. For our purposes this challenge can be overcome by the following useful lemma:

Lemma 5.6.5. *Let $\mathbf{y} \in Y$ such that $\psi(\mathbf{y})$ is in the interior of Gröbner cone $\mathcal{K}_i$. Then*

$$\text{argmin}\{\psi(\mathbf{y}) \cdot \mathbf{x} : A\mathbf{x} \leq \pi(\mathbf{z}), \mathbf{x} \in X\} = \{\mathbf{x}_i(\mathbf{z}')\} \tag{5.26}$$

from some $\mathbf{z}' \in Z$. Recall, that $\mathbf{x}_i(\mathbf{z}')$ is the unique optimal solution to the equality-form integer program $IP_{\psi(\mathbf{y})}(\pi(\mathbf{z}'))$. In particular, the set of optimal solutions to the family of

inequality constrained in integer programs is $\mathcal{O}_i$, the set of solutions to the corresponding family of equality constrained integer programs $IP_{\mathbf{c}_i}$ where $\mathbf{c}_i \in \operatorname{relint} \mathcal{K}_i$.

Proof. Suppose $\mathbf{x}^*$ is in the $\operatorname{argmin}\{\psi(\mathbf{y}) \cdot \mathbf{x} : A\mathbf{x} \leq \pi(\mathbf{z}), \mathbf{x} \in X\}$. Then $\mathbf{x}^*$ is also in the set

$$\operatorname{argmin}\{\psi(\mathbf{y}) \cdot \mathbf{x} : A\mathbf{x} = A\mathbf{x}^*, \mathbf{x} \in X\}$$

and thus $\mathbf{x}^* \in \mathcal{O}_i$. Since when $\psi(\mathbf{y})$ is generic it follows immediately that

$$\operatorname{argmin}\{\psi(\mathbf{y}) \cdot \mathbf{x} : A\mathbf{x} \leq \pi(\mathbf{z}'), \mathbf{x} \in X\} = \{\mathbf{x}_i(A\mathbf{x}^*)\}.$$

By assumption (A2), we define $\mathbf{z}' = A\mathbf{x}^* \in Z$. □

This lemma and assumption (A2) taken together establish a link between the equality and inequality cases, a link that will be exploited in following key result of this section:

Theorem 5.6.6 (Existence Theorem for Inequality Constraints)**.** *Consider an inequality constrained bilevel optimization problem* (5.3)*. Given assumptions (A1)-(A5) an optimal solution exists.*

As the proof is rather involved we have provided the following brief synopsis, which has several important steps.

Step 1. First we fix a Gröbner cone $\mathcal{K}_i$ and consider finding an optimal choice of $\mathbf{x}$, which we denote by $\mathbf{x}_i$. We move to finding an optimal choice of $\mathbf{z}$, which we denote by $\mathbf{z}_i$, that will ensure the follower chooses $\mathbf{x}_i$.

Step 2. Show that when $\mathbf{y} \in Y$ is fixed such that $\psi(\mathbf{y})$ is in the interior of $\mathcal{K}_i$ an optimal choice for the pair $(\mathbf{x}, \mathbf{z})$ is indeed $(\mathbf{x}_i, \mathbf{z}_i)$.

Step 3. We consider arbitrary $\mathbf{y}$ and use the Convex Hull Theorem 5.4.2 to show that an optimal choice for the pair $(\mathbf{x}, \mathbf{z})$ for a fixed arbitrary $\mathbf{y}$ is $(\mathbf{x}_i, \mathbf{z}_i)$ for some $i \in I(\psi(\mathbf{y}))$.

Step 4. Show the existence of an optimal choice for $\mathbf{y}$.

Proof. **Step 1.** Fix a Gröbner cone $\mathcal{K}_i$ and let $\mathcal{P}_i$ denote the set of feasible $\mathbf{x}$'s in $\mathcal{O}_i$. That is,

$$\mathcal{P}_i = \{\mathbf{x} \in \mathcal{O}_i : A\mathbf{x} \in \pi^{\leq}\} \cap X.$$

Now, for every $\mathbf{x} \in \mathcal{P}_i$ choose a vector

$$\zeta(\mathbf{x}) \in \text{argmin}_{\mathbf{z}}\{f(\mathbf{x}, \mathbf{z}) : (\mathbf{x}, \mathbf{z}) \in P, A\mathbf{x} \leq \pi(\mathbf{z})\}.$$

Assumption (A1) implies that this infimum is indeed attained. Next choose

$$\mathbf{x}_i \in \text{argmin}_{\mathbf{x}}\{f(\mathbf{x}, \zeta(\mathbf{x})) : \mathbf{x} \in \mathcal{P}_i\} \tag{5.27}$$

which is possible for the same reasoning as in the equality case ($\mathbf{x}$ can be chosen from a bounded discrete set: see the proof of the Existence Theorem 5.6.3).

By assumption (A2) there exists $\mathbf{z}_i \in Z$ such that $\pi(\mathbf{z}_i) = A\mathbf{x}_i$. We now claim that $\mathbf{x}_i$ and $\mathbf{z}_i$ are compatible; that is, if the leader chooses $\mathbf{z}_i$ then she can ensure the follower chooses $\mathbf{x}_i$. The reasoning is as follows: for every $\mathbf{y} \in Y$ and for every $\mathbf{x} \in X$ such that $A\mathbf{x} \leq \pi(\mathbf{z}_i)$ assumption (A4) implies

$$\begin{aligned} \psi(\mathbf{y}) \cdot \mathbf{x} &= -\mathbf{u}_{\mathbf{y}} A\mathbf{x} \geq -\mathbf{u}_{\mathbf{y}} \pi(\mathbf{z}_i) \\ &= -\mathbf{u}_{\mathbf{y}} A\mathbf{x}_i \\ &= \psi(\mathbf{y}) \cdot \mathbf{x}_i. \end{aligned}$$

That is, $\mathbf{x}_i$ is an optimum solution to the inequality constrained follower's problem

$$\min_{\mathbf{x}}\{\psi(\mathbf{y}) \cdot x : A\mathbf{x} \leq \pi(\mathbf{z}_i), \mathbf{x} \in X\}$$

and so by the optimistic assumption (that is, the leader can choose any $\mathbf{x}$ from among the

alternate of the follower) $\mathbf{x}_i$ and $\mathbf{z}_i$ are compatible.

Step 2. Next show $(\mathbf{x}_i, \mathbf{z}_i)$ is indeed an optimal choice of $(\mathbf{x}, \mathbf{z})$ when $\psi(\mathbf{y})$ is in the interior of $\mathcal{K}_i$. Since $\pi(\mathbf{z}_i) = A\mathbf{x}_i$ it follows $\pi(\mathbf{z}_i) \leq \pi(\zeta(\mathbf{x}_i))$ and by assumption (A3)

$$f(\mathbf{x}, \mathbf{z}_i) \leq f(\mathbf{x}, \zeta(\mathbf{x}_i)). \tag{5.28}$$

Thus, for every $\mathbf{x} \in \mathcal{P}_i$ and every $\mathbf{z} \in Z$ such that $A\mathbf{x} \leq \pi(\mathbf{z})$:

$$\begin{aligned} f(\mathbf{x}, \mathbf{z}) &\geq f(\mathbf{x}, \zeta(\mathbf{x})) && (5.29)\\ &\geq f(\mathbf{x}_i, \zeta(\mathbf{x}_i)) && (5.30)\\ &\geq f(\mathbf{x}_i, \mathbf{z}_i) && (5.31) \end{aligned}$$

From (5.29) we use the definition of $\zeta(\mathbf{x})$. From (5.30) to (5.31) we use the definition of $\mathbf{x}_i$. The final step follows by (5.28).

We can therefore conclude that for $\mathbf{y} \in Y$ such that $\psi(\mathbf{y})$ is in the interior of K_i:

$$\begin{aligned} \theta(\mathbf{y}) &\doteq \inf_{\mathbf{x},\mathbf{z}} \left\{ f(\mathbf{x}, \mathbf{z}) : (\mathbf{x}, \mathbf{z}) \in P, \mathbf{x} \in \operatorname{argmin}\{\psi(\mathbf{y}) \cdot \mathbf{x} : A\mathbf{x} \leq \pi(\mathbf{z}), \mathbf{x} \in X\} \right\} && (5.32)\\ &= \inf_{\mathbf{x},\mathbf{z}} \left\{ f(\mathbf{x}, \mathbf{z}) : A\mathbf{x} \leq \pi(\mathbf{z}), \mathbf{x} \in \mathcal{P}_i, (\mathbf{x}, \mathbf{z}) \in P \right\} && (5.33)\\ &= f(\mathbf{x}_i, \mathbf{z}_i) && (5.34) \end{aligned}$$

In words, for any $\mathbf{y}$ such that $\psi(\mathbf{y})$ is in the interior of $\mathcal{K}_i$ an optimal choice of $\mathbf{x}$ and $\mathbf{z}$ exists and can be taken as $\mathbf{x}_i$ and $\mathbf{z}_i$ respectively.

Step 3. Now, consider an arbitrary choice of $\mathbf{y} \in Y$. We will show

$$\theta(\mathbf{y}) = \min_{i \in I(\psi(\mathbf{y}))} \left\{ f(\mathbf{x}_i, \mathbf{z}_i) \right\} \tag{5.35}$$

Suppose there exists a $\mathbf{z} \in Z$ such that $\mathbf{x} \in \text{argmin}\{\psi(\mathbf{y}) \cdot \mathbf{x} : A\mathbf{x} \leq \pi(\mathbf{z}), \mathbf{x} \in X\}$. By assumption (A2) there exists $\mathbf{z}' \in \{\mathbf{z} \in Z \text{ such that } A\mathbf{x} = \pi(\mathbf{z}')\}$, and it follows

$$x \in \text{argmin}\{\psi(\mathbf{y}) \cdot \mathbf{x} : A\mathbf{x} = \pi(\mathbf{z}'), \mathbf{x} \in X\}.$$

By the Convex Hull Theorem:

$$\mathbf{x} \in \text{conv}\{\mathbf{x}_i(\pi(\mathbf{z}')) : i \in I(\psi(\mathbf{y}))\} \cap X.$$

Now, since f is quasiconcave the optimal value of

$$\min_{\mathbf{x}} \left\{ f(\mathbf{x}, z') : \mathbf{x} \in \text{conv}\{\mathbf{x}_i(\mathbf{z}') : i \in I(\psi(\mathbf{y}))\} \cap X \right\}$$

occurs at an extreme point of the feasible region. Hence, for any $\mathbf{y} \in Y$ and for any $\mathbf{x}$ and $\mathbf{z}$ such that $\mathbf{x} \in \text{argmin}\{\psi(\mathbf{y}) \cdot \mathbf{x} : A\mathbf{x} \leq \pi(\mathbf{z}), \mathbf{x} \in X\}$:

$$\begin{aligned} f(\mathbf{x}, \mathbf{z}) &\geq \left(\min_{i \in I(\psi(\mathbf{y}))} f(\mathbf{x}_i(\mathbf{z}'), \mathbf{z}), \right) && (5.36) \\ &\geq \min_{i \in I(\psi(\mathbf{y}))} \left\{ f(\mathbf{x}_i(\mathbf{z}'), \mathbf{z}') \right. && (5.37) \\ &\geq \min_{i} f(\mathbf{x}_i, \mathbf{z}_i) && (5.38) \end{aligned}$$

From (5.36) to (5.37) we use the fact that $\pi(\mathbf{z}') = A\mathbf{x} \leq \pi(\mathbf{z})$ and assumption (A3). The last step follows f (5.31). This in turn will imply (5.35).

Step 4. By the previous steps we have shown that the objective value of the inequality constrained problem as a function of $\mathbf{y}$ can be expressed as:

$$\begin{aligned} F(\mathbf{y}) &= \theta(\mathbf{y}) + g(\mathbf{y}) \\ &= \min_{i \in I(\psi(\mathbf{y}))} \left\{ f(\mathbf{x}_i, \mathbf{z}_i) + g(\mathbf{y}) \right\} \end{aligned}$$

and thus the optimal value is

$$\inf_{\mathbf{y}\in Y} F(\mathbf{y}) = \min_{i\in I(\psi(\mathbf{y}))} \left\{ f(\mathbf{x}_i, \mathbf{z}_i) + \inf_{\mathbf{y}\in Y(i)} g(\mathbf{y}) \right\} \tag{5.39}$$

by the same reasoning found in the proof of the Existence Theorem 5.6.3. Assumption (A5) implies that the ith cone subproblem $\inf_{\mathbf{y}\in Y(i)} g(\mathbf{y})$ has an attainable infimum, call it $\mathbf{y}_i$.

We can conclude that if

$$j \in \operatorname{argmin}_{i\in I(\psi(\mathbf{y}))} \left\{ f(\mathbf{x}_i, \mathbf{z}_i) + g(\mathbf{y}) \right\}$$

an optimal solution to the inequality constrained problem (5.3) exists and is equal to $(\mathbf{x}_j, \mathbf{y}_j, \mathbf{z}_j)$. □

In fact more can be said about the nature of these optimal solutions. Assumption (A2) in some sense turns the problem into an equality constrained problem, at least as far as finding optimal solutions is concerned. The following corollary makes this precise.

Corollary 5.6.7. *Let* $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ *be an optimal solution to an instance of the equality constrained problem* (5.11). *Then* $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ *is an optimal solution to the instance of the inequality constrained problem* (5.3) *with the same data.*

Proof. By the Decomposition Theorem there exists a $j \in \operatorname{argmin}_{i\in I}\{\xi_i + \min_{y\in Y(i)} g(\mathbf{y})\}$ such that the optimal solution, $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$, has the following properties:

- $(\mathbf{x}^*, \mathbf{z}^*)$ solves the jth fixed-cost bilevel subproblem:

$$\xi_j = \min_{\mathbf{x},\mathbf{z}} \{ f(\mathbf{x}, \mathbf{z}) : (\mathbf{x}, \mathbf{z}) \in P, \mathbf{x} \in \operatorname{argmin}_{\mathbf{x}\in X} \{ \mathbf{c}\cdot\mathbf{x} : A\mathbf{x} = \pi(\mathbf{z}) \} \} \tag{5.40}$$

  for some fixed unit cost $\mathbf{c} \in \operatorname{relint} \mathcal{K}_j$.

- $\mathbf{y}^*$ is an optimal solution to the jth cone optimization subproblem: $\min_{\mathbf{y}\in Y(j)} g(\mathbf{y})$

We claim that $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*)$ is an optimal solution to the inequality constrained problem with the same data. In fact, we will show $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*) = (\mathbf{x}_j, \mathbf{y}_j, \mathbf{z}_j)$ where $(\mathbf{x}_j, \mathbf{y}_j, \mathbf{z}_j)$ is a solution to inequality constrained problem in notation of the previous proof.

To begin, note that $\mathbf{x}^* \in \mathcal{P}_j$. Thus by (5.31):

$$f(\mathbf{x}^*, \mathbf{z}^*) \geq f(\mathbf{x}_j, \mathbf{z}_j)$$

Thus (x_j, z_j) is an optimal solution to (5.40). Hence we can take $(\mathbf{x}^*, \mathbf{z}^*) = (\mathbf{x}_j, \mathbf{z}_j)$ since we are assuming the optimistic approach.

The jth cone optimization subproblems are exactly the same in both the inequality and equality constrained cases. It follows that we can take $\mathbf{y}^* = \mathbf{y}_j$. Thus, $(\mathbf{x}^*, \mathbf{y}^*, \mathbf{z}^*) = (\mathbf{x}_j, \mathbf{y}_j, \mathbf{z}_j)$ is an optimal solution to the inequality constrained problem. □

To conclude this section we give some brief justification for the assumptions (A1)-(A5). Assumption (A1) is in fact weaker than the required assumptions for the equality constrained problem, since π is required only to be upper semicontinuous and not continuous as in the statement of the Existence Theorem 5.6.3.

Assumption (A3) is reasonable when we consider a possible application of our discrete bilevel optimization problem. Suppose the leader is a principal who can choose $\mathbf{y}$ and $\mathbf{z}$ to influence the payment ($\psi(\mathbf{y})$) and available resources ($\pi(\mathbf{z})$) that face her agent, the follower, who takes some action $\mathbf{x}$ that is of benefit to both the principal and the agent. Viewed in this way, the comonotonicity described in (A3) is reasonable since providing more resources (an increase $\pi(\mathbf{z})$) should cost the principal more (an increase in $f(\mathbf{x}, \mathbf{z})$ for a fixed $\mathbf{x}$). Note that the problem in Example 5.6.4 violates assumption (A3).

As was mentioned above, (A2) is the key assumption that creates a strong connection between the equality and inequality cases and it can be seen to drive the result of Corollary 5.6.7. Again taking the principal-agent interpretation above, one could assume that if some

amount of resource $\pi(\mathbf{z})$ could be provided by the principal, then any amount less than $\pi(\mathbf{z})$ and greater than or equal zero could also be provided. In addition if A has positive entries, which it most likely would in the case of resource utilization problems we have described, then $A\mathbf{x} \geq 0$ and hence (A2) seems a reasonable assumption.

Lastly, assumption (A4) is the most difficult to justify. There seems no easy interpretation as to why the cost vector ought to be from the row space of the matrix $-A$. In fact, finding alternate assumptions that achieve the function of (A4) in the proofs of the theorems in this section is an area for further exploration.

5.7 Summary and directions for further research

In this chapter we explored how the structure of consistent fans can decompose bilevel optimization problems into a finite set of subproblems with fixed-unit cost linear lower level objectives. Using this decomposition we also present algorithms for solving special cases and sufficient conditions for the existence of optimal solutions when some of the variables are continuous.

There are some major weaknesses of this analysis, each of which we hope can motivate further investigation. The most glaring weakness is that in general the number of subproblems created through decomposition can be exponential in the input size, even in fixed dimension. This gives a strong indication these methods are impractical in a general setting. Thus, an important area for further research is to is to search for special cases where we might limit the number of subproblems. As we saw, k-fold bilevel integer programming is one such case, and we believe there is potential in finding other cases.

In addition, there are some potential ways we can avoid solving each subproblem. Developing branching algorithms may give exact or approximate solutions by only exploring a subset of the subproblems. The existing algorithms for computing the Gröbner cones of a

matrix A have the attractive feature that they start with a single cone and compute adjacent cones through local computations (see for instance [45]). This means that the all of the Gröbner cones need not be computed as once, and thus there is the potential, using bounds, that some cones need not be explored.

A second computational challenge is in actually solving each of the fixed unit-cost subproblems. Although there are different methodologies for solving bilevel integer programs ([32, 61]) to date there are no publicly available implementations of these algorithms. Clearly, the effectiveness of the overall algorithm depends on our ability to solve each subproblem effectively.

Lastly, we presented some sufficient conditions for the existence of solutions to bilevel mixed-integer programs and we feel that these conditions could possibly be tightened and improved with further investigation.

Bibliography

[1] 4ti2 team. 4ti2 – a software package for algebraic, geometric and combinatorial problems on linear spaces. Available at www.4ti2.de.

[2] C. Àlvarez, J. Gabarró, and M. Serna. Pure Nash equilibria in games with a large number of actions. In *MFCS: Proceedings of the International Symposium on Mathematical Foundations of Computer Science*, pages 95–106, 2005.

[3] E. Anshelevich, A. Dasgupta, J. Kleinberg, E. Tardos, T. Wexler, and T. Roughgarden. The price of stability for network design with fair cost allocation. In *FOCS: Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*, pages 295–305, 2004.

[4] S. Arora and B. Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[5] J. F. Bard and J. Moore. An algorithm for the discrete bilevel progamming problem. *Naval Research Logistics*, 39:419–435, 1992.

[6] A. Barvinok. A polynomial time algorithm for counting integral points in polyhedra when the dimension is fixed. *Mathematics of Operations Research*, 19(4):769–779, 1994.

[7] A. Barvinok. Computing the Ehrhart quasi-polynomial of a rational simplex. *Mathematics of Computation*, 75(255):1449–1466, 2006.

[8] A. Barvinok and K. M. Woods. Short rational generating functions for lattice point problems. *Journal of the American Mathematical Society*, 16(957-979), 2003.

[9] M. Beck and S. Robbins. *Computing the Continuous Discretely: Integer-point Enumeration in Polyhedra*. Springer, 2007.

[10] N. A. R. Bhat and K. Leyton-Brown. Computing Nash equilibria of action-graph games. In *UAI: Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 35–42, 2004.

[11] R. Bhattacharjee, F. Thuijsman, and O. J. Vrieze. Polytope games. *Journal of Optimization Theory and Applications*, 105(3):567–588, 2000.

[12] C. Borgs, J. Chayes, N. Immorlica, A. T. Kalai, V. Mirrokni, and C. Papadimitriou. The myth of the folk theorem. In *STOC: Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 365–372, 2008.

[13] F. Brandt, F. Fischer, and M. Holzer. Symmetries and the complexity of pure Nash equilibrium. *Journal of Computer and System Sciences*, 75(3):163–177, 2009.

[14] G.P. Cachon and S. Netessine. Game theory in supply chain analysis. In S. D. Wu D. Simchi-Levi and Z. Shen, editors, *Handbook of Quantitative Supply Chain Analysis: Modeling in the eBusiness Era*, chapter 2, pages 13–65. Kluwer, 2004.

[15] X. Chen and X. Deng. Settling the complexity of two-player Nash equilibrium. In *FOCS: Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*, pages 261–272, 2006.

[16] S.F. Cheng, D.M. Reeves, Y. Vorobeychik, and M.P. Wellman. Notes on equilibria in symmetric games. In *AAMAS-04 Workshop on Game-Theoretic and Decision-Theoretic Agents*, 2004.

[17] A. Chinchuluun, P.M. Pardalos, and H.X. Huang. Multilevel(hierarchical) optimization: Complexity issues, optimality conditions, algorithms. In *Advances in Applied Mathematics and Global Optimization in Honor of Gilbert Strang (Advances in Mechanics and Mathematics Series Volume 17)*, pages 197–222. Springer, 2009.

[18] B. Colson, P. Marcotte, and G. Savard. An overview of bilevel optimization. *Annals of Operations Research*, 153(1):235–256, 2007.

[19] P. Conti and C. Traverso. Buchberger algorithm and integer programming. In *AAECC: Proceedings of the International Symposium, on Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pages 130–139, 1991.

[20] J-P. Cote, P. Marcotte, and G. Savard. A bilevel modelling approach to pricing and fare optimization in the airline industry. *Journal of Revenue and Pricing Management*, 2(1):23–47, 2003.

[21] C. Daskalakis and C. H. Papadimitriou. Three-player games are hard. In *ECCC: Proceedings of the Electronic Colloquium on Computational Complexity*, 2005, TR05-139.

[22] J. A. De Loera, D. Haws, R. Hemmecke, P. Huggins, and R. Yoshida. Three kinds of integer programming algorithms based on barvinok's rational functions. In *IPCO*, pages 244–255, 2004.

[23] J. A. De Loera, R. Hemmecke, and M. Köppe. Pareto optima of multicriteria integer linear programs. *INFORMS Journal on Computing*, 21(1):39–48, 2009.

[24] J. A. De Loera, R. Hemmecke, M. Köppe, and R. Weismantel. FPTAS for optimizing polynomials over the mixed-integer points of polytopes in fixed dimension. *Mathematical Programmming*, 115(2):273–290, 2008.

[25] J. A. De Loera, R. Hemmecke, S. Onn, and R. Weismantel. N-fold integer programming. *Discrete Optimization*, 5(2):231–241, 2008.

[26] J. A. De Loera, R. Hemmecke, J. Tauzer, and R. Yoshida. Effective lattice point counting in rational convex polytopes. *Journal of Symbolic Computation*, 38(4):1273–1302, 2004.

[27] S. Dempe. Discrete bilevel optimization problems. Preprint, can be accessed at http://citeseer.ist.psu.edu/dempe01discrete.html, 2001.

[28] S. Dempe. *Foundations of bilevel programming*. Kluwer, 2002.

[29] S. Dempe, V. Kalashnikov, and R.Z. Rıos-Mercado. Discrete bilevel programming: Application to a natural gas cash-out problem. *European Journal of Operational Research*, 166(2):469–488, 2005.

[30] S. Dempe and K. Richter. Bilevel programming with knapsack constraints. *Central European Journal of Operations Research*, 8:93–107, 2000.

[31] S. DeNegre and T. Ralphs. Multiobjective mixed-integer Stackelberg games. International symposium on mathematical programming, 2006.

[32] S. T. DeNegre and T. K. Ralphs. A branch-and-cut algorithm for integer bilevel linear programs. In *Operations Research and Cyber-Infrastructure*, pages 65–78. Springer, 2009.

[33] X. Deng. Complexity issues in bilevel linear programming. In A. Migdalas, P. M. Pardalos, and V. Varbrand, editors, *Multilevel Optimization: Algorithms and Applications*, chapter 6, pages 149–164. Springer, 1998.

[34] N.R. Devanur and R. Kannan. Market equilibria in polynomial time for fixed number of goods or agents. In *FOCS: Proceedings of the Annual IEEE Symposium on Foundations of Computer Science*, pages 45–53, 2008.

[35] J. Dunkel and A. Schulz. On the compexity of pure-strategy Nash equilibria in congestion and local-effect games. *Mathematics of Operations Research*, 33(4):851–868, 2008.

[36] H. Edelsbrunner, J. O'Rouke, and R. Seidel. Constructing arrangements of lines and hyperplanes with applications. *SIAM Journal of Computing*, 15(2):341–363, 1986.

[37] F. Eisenbrand and G. Shmonin. Parametric integer programming in fixed dimension. *Mathematics of Operations Research*, 33(4):839–850, 2008.

[38] E. Erkut and F. Gzara. Solving the hazmat transport network design problem. *Computers and Operations Research*, 35(7):2234–2247, 2008.

[39] A. Fabrikant, C. Papadimitriou, and K. Talwar. The complexity of pure Nash equilibria. In *STOC: Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 604–612, 2004.

[40] K. Fischer. Sequential discrete p-facility models for competitive location planning. *Annals of Operations Research*, 111:253–270, 2004.

[41] G. Gottlob, G. Greco, and F. Scarcello. Pure Nash equilibria: Hard and easy games. *Journal of Artificial Intelligence Research*, 24:357–406, 2005.

[42] J. E. Graver. On the foundations of linear and integer linear programming I. *Mathematical Programming*, 9(1):207–226, 1975.

[43] Z. H. Gümüş and C. A. Fluodas. Global optimization of mixed-integer bilevel programming problems. *Computational Management Science*, 2(3):181–212, 2005.

[44] R. Hemmecke, S. Onn, and R. Weismantel. Nash-equilibria and N-fold integer programming. *Arxiv preprint arXiv:0903.4577*, 2009.

[45] B. Huber and R. R. Thomas. Computing Gröbner fans of toric ideals. *Experimental Mathematics*, 2000(3):321–331, 9.

[46] R-H. Jan and M-S. Chern. Nonlinear integer bilevel programming. *European Journal of Operational Research*, 72(3):574–587, 1994.

[47] A. N. Jensen. CaTS version 2.2: A user's manual. Available at `http://www.soopadoopa.dk/anders/cats/cats.html`, 2003.

[48] A. N. Jensen. *Algorithm aspects of Gröbner fans and tropical varieties*. PhD thesis, University of Aarhus, July 2007.

[49] R. Kannan. Lattice translates of a polytope and the Frobenius problem. *Combinatorica*, 12(2):161–177, 1992.

[50] M. Kearns, M. Littman, and S. Singh. Graphical models for game theory. In *UAI: Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 253–260, 2001.

[51] M. Köppe. `LattE machiatto`. Available from URL `http://www.math.ucdavis.edu/~mkoeppe/latte/`, 2006.

[52] M. Köppe. LattE machiatto benchmarks. `http://www.math.ucdavis.edu/~mkoeppe/latte/latte-benchmark-2007-11-01.html`, 2007.

[53] M. Köppe, S. Verdoolaege, and K. M. Woods. An implementation of the Barvinok-Woods integer projection algorithm. In *International Conference on Information Theory and Statistical Learning*, pages 53–59, 2008.

[54] E. Koutsoupias and C. Papadimitriou. Worst-case equilibria. *Computer Science Review*, 3(2):65–69, 2009.

[55] S. Kwek and K. Melhorn. Optimal search for rationals. *Information Processing Letters*, 86(1):23–26, 2003.

[56] M. Labbé, P. Marcotte, and G. Savard. A bilevel model of taxation and its application to optimal highway pricing. *Management Science*, 44(12):1608–1622, 1998.

[57] E.L. Lawler and D.E. Wood. Branch-and-bound methods: A survey. *Operations Research*, 14(4):699–719, 1966.

[58] H. W. Lenstra Jr. Integer programming with a fixed number of variables. *Mathematics of Operations Research*, 8(4):538–548, 1983.

[59] K. Leyton-Brown and M. Tennenholtz. Local-effect games. In *IJCAI: Proceedings of the International Joint Conference on Artificial Intelligence*, pages 772–780, 2003.

[60] D. Monderer and L. S. Shapley. Potential games. *Games and Economic Behavior*, 14:124–143, 1996.

[61] J. Moore and J. Bard. The mixed integer linear bilevel programming problem. *Operations Research*, 38:911–921, 1990.

[62] R.E. Moore. Global optimization to prescribed accuracy. *Computers & Mathematics with Applications*, 21(6-7):25–39, 1991.

[63] J. Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286–295, February 1951.

[64] J. Von Neumann and M. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944.

[65] T. Nguyen and É. Tardos. Approximately maximizing efficiency and revenue in polyhedral environments. In *ACM-EC: Proceedings of the ACM Conference on Electronic Commerce*, pages 11–19, 2007.

[66] N. Nisan, T. Roughgarden, É. Tardos, and V. V. Vazirani. *Algorithmic Game Theory.* Cambridge University Press, 2007.

[67] C. Papadimitriou. The complexity of finding Nash equilibria. In N. Nisan, T. Roughgarden, E. Tardos, and V. V. Vazirani, editors, *Algorithmic Game Thoery*, chapter 2, pages 29–52. Cambridge University Press, 2007.

[68] C. H. Papadimitriou and T. Roughgarden. Computing correlated equilibria in multiplayer games. *Journal of the ACM*, 55(3):1–29, 2008.

[69] C.H. Papadimitriou and M. Yannakakis. On complexity as bounded rationality. In *STOC: Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 726–733, 1994.

[70] R.W. Rosenthal. A class of games possessing pure-strategy Nash equilibria. *International Journal of Game Theory*, 2:65–67, 1973.

[71] T. Roughgarden. *Selfish routing and the price of anarchy.* The MIT Press, 2005.

[72] K. H. Sahin and A. R. Ciric. A dual temperature simulated annealing approach for solving bilevel programming problems. *Computers and Chemical Engineering*, 23(1):11–25, 1998.

[73] G. Schoenebeck and S. Vadhan. The computational complexity of Nash equilibria in concisely represented games. In *ACM-EC: Proceedings of the ACM Conference on Electronic Commerce*, pages 270–279, 2006.

[74] A. Schrijver. *Theory of linear and integer programming.* Wiley, 1986.

[75] S. Schwarze. *Path Player Games.* Springer, 2009.

[76] J.C. Smith, C. Lim, and A. Alptekinoglu. New product introduction against a predator: A bilevel mixed-integer programming approach. *Naval Research Logistics*, 56(8):714–729, 2009.

[77] L. J. Stockmeyer. The polynomial-time hierarchy. *Theoretical Computer Science*, 3:1–22, 1976.

[78] B. Sturmfels. *Gröbner Bases and Convex Polytopes*. American Mathematical Society, 1996.

[79] B. Sturmfels and R. R. Thomas. Variation of cost functions in integer programming. *Mathematical Programming*, 77(3):357–387, 1997.

[80] R. R. Thomas. A geometric Buchberger algorithm for integer programming. *Mathematics of Operations Research*, 20:864–884, 1995.

[81] R. R. Thomas. The structure of group relaxations. In K. Aardal, G. Nemhauser, and R. Weismantel, editors, *Discrete Optimization*, volume 12 of *Handbooks in Operations Research and Management Science*. Elsevier, 2005.

[82] S. Verdoolaege. `barvinok`. Available from URL `http://freshmeat.net/projects/barvinok/`, 2006.

[83] L. Vicente, G. Savard, and J. Judice. Discrete linear bilevel programming problem. *Journal of Optimization Theory and Applications*, 89(3):597–614, June 1996.

[84] H. von Stackelberg. *Marktform und Gleichgewicht*. Springer-Verlag, 1934. English translation entitled as Market Structure and Equilibrium, published in 1952 by Oxford University Press.

[85] U. P. Wen and Y. H. Yang. Algorithms for solving the mixed integer two-level linear programming problem. *Computers and Operations Research*, 17(2):133–142, 1990.

[86] V. A. Zalgaller. Representations of functions of several variables by difference of convex functions. *Journal of Mathematical Sciences*, 100(3):2209–2227, 2000.

[87] G. M. Ziegler. *Lectures on Polytopes*, volume 152. Springer-Verlag, New York, 1995.